

## Web Technology

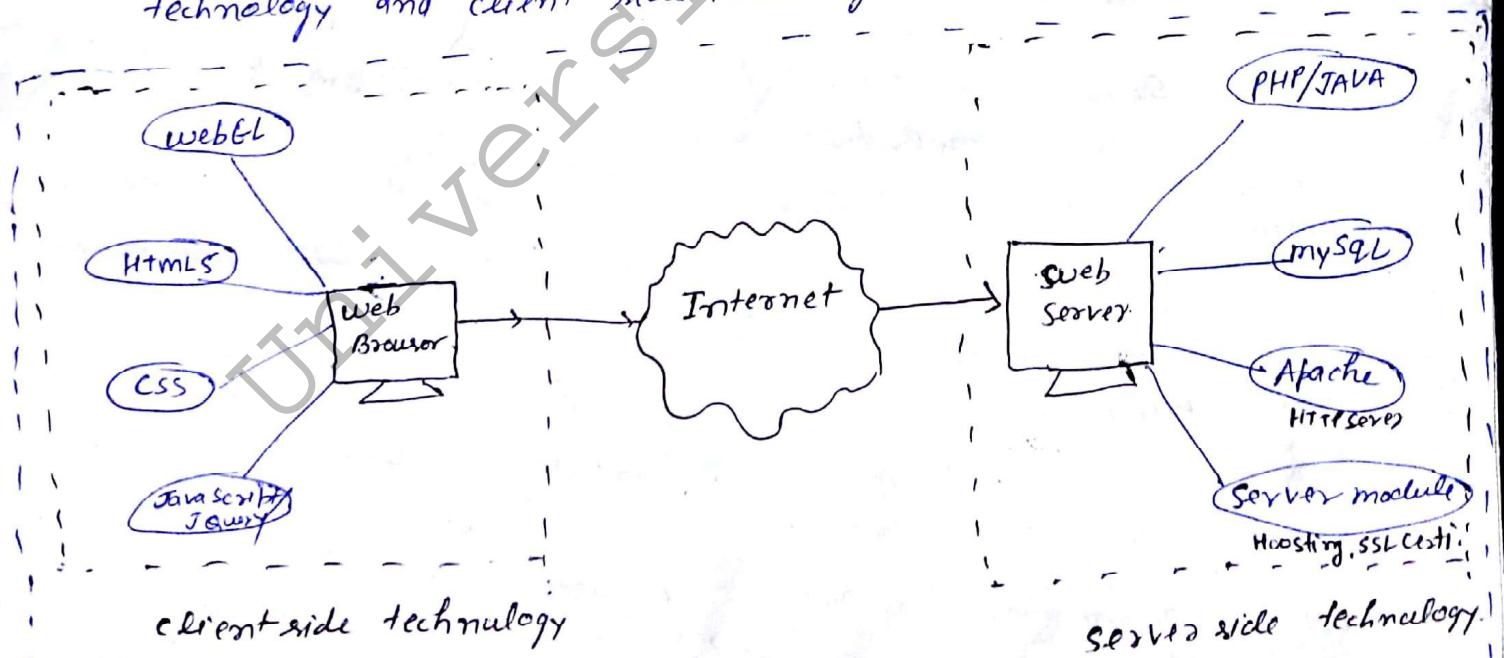
### UNIT - I

Introduction: Computers don't communicate each other like people do. Instead computers requires codes and commands. These binary code and command allow computers to process needed information. Every second billions upon billions of ones and zeros are processed in order to provide you information.

The method by which computers communicate with each other through the use of markup language and multimedia is known as web technology.

Web technology is a collective name for technology primarily for the word wide web. The focus of web technology is the creation, maintenance and development of web-based Application.

The web technology is the combination of server-side technology and client side technology.



1. Desktop Application
2. Web application
3. Native Application

## Web Terminology

1. web: collection of electronic resources is called web. Tim Berners-Lee developed first web browser in 1990.
2. w3c: The world wide web consortium (w3c) is an international community where member organization, a full-time staff, and public works together to develop web standard.
3. Network: A network is collection of computers, servers, mainframe, or other devices connected to one another allow the sharing of data.
4. Internet: The Internet is the global system of interconnected computer networks that use the Internet protocol (TCP/IP) to link device world wide. Vint Cerf father of Internet.
5. E-mail.: Electronic mail services. first time introduced by an American and after web mail developed by an Indian that is hotmail.com developed by Sabeer Bhatia from chandigarh.
6. SMTP: Simple mail transfer Protocol is an internet standard for electronic mail transmission.
7. MIME: multipurpose Internet mail <sup>Extending.</sup> exchange is an internet standard that extends the format of email to support: Text, audio, video, image etc.
8. FTP: File transfer Protocol is a standard network protocol used for the transfer computer file between client and server on a computer Network.
9. TELNET: Telecommunication Network is used to connect computers. local host to Remote host, with the help of TELNET we can log into in remote system with the help of local host.  
e.g Teamviewer.
10. HTTP: Hyper text transfer Protocol is taking HTML Pages and it's rendering HTML Pages

11. Blog : A regularly updated website or web page. Typically run by an individual or small group. On a blog every update first displayed is reverse chronological order.

12. Forum : online discussion website.

13. Addon : Additional features to the web browser e.g. ebay toolbar, games toolbar.

14. Plugin : it is a kind of <sup>Additional</sup> software to the browser e.g. flash player.

15. Webpages : it is page open with the help of web browsers. and that page is developed in HTML. There are two type of webpages. static and dynamic.

└ HTML + CSS

└ HTML + CSS + Javascript.

16. Website : Collection of webpages is called website.  
static website → only on client level technology  
dynamic → developed on client and server side technology.

### History of Internet and Web

The internet was created in Jan. 2, 1969 by Advanced Research Project Agency (ARPA) which gave its first name THE ARPANET. The ARPANET users started sharing the resource of the computer by using TELNET protocol.

In 1975, Vint Cerf developed a transport layer protocol IP and also developed Network Protocol (IP) at Network layer.

In 1980 the National Science Foundation (NSF) founded the TCP/IP Network to establish a network, (NSFNET). In 1988, traffic increased so quickly introduce T3 lines which are 45 mbps lines.

In 1991, The NSF established a new network that is known as NREN (National Research and Educational Network).  
 In 1989 Tim Berners Lee proposed concept of www.  
 In 1992 www was introduced by ~~NEC~~ Tim Berners Lee (British computer scientist) he does this by bringing three main element.

- i. HTTP
- ii. HTML
- iii. URL (Uniform Resource Locator)

### Protocol Governing Web



1. Application Layer : - HTTP, FTP, TELNET, SMTP, IMAP, POP, DNS  
SSL
2. Transport Layer : TCP, UDP
3. Network layer : IP, ARP, ICMP  
Internetlayer

#### I. Application Layer Protocol

(i) HTTP : - The Hypertext Transfer protocol used by www and this protocol define how message are formed and transmitted.

HTTP functions web client as a request-response protocol in client-server computing model.

Request

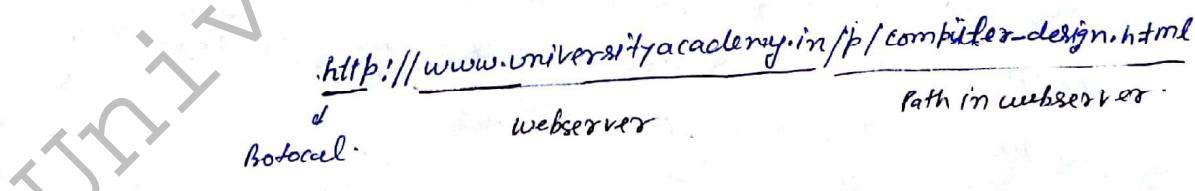
IE, chrome

PROTOCOL

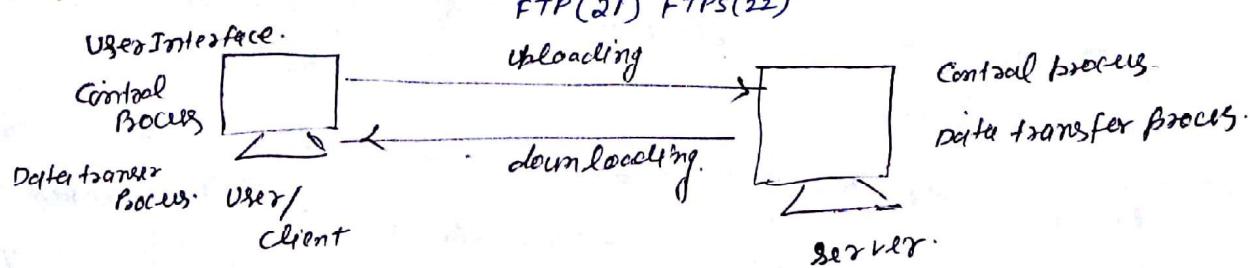
Response.

Application server

HTML, PDF etc.



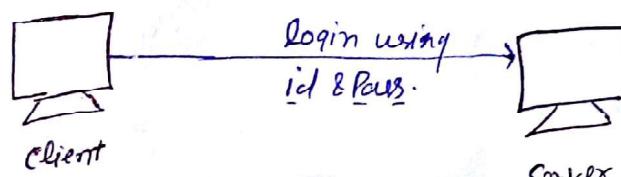
#### (ii) FTP !



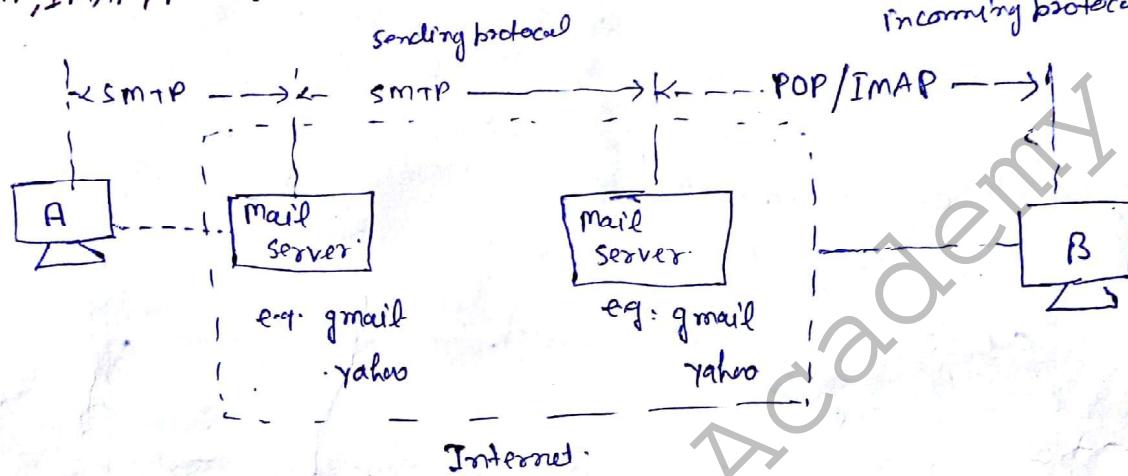
e.g. Dropbox

File transfer protocol is a standard Internet protocol for transmitting files between two computer on the Internet over TCP/IP connection.

(iii) Telnet: (Putty, Teamviewer) : Telnet is a ~~text~~ protocol that allows you to connect to remote computer over TCP/IP network.



(iv) SMTP, IMAP, POP : SMTP is the standard protocol for send and receive emails. incoming protocol.



POP3  
(Post office Protocol)  
1. does not synchronize folder.

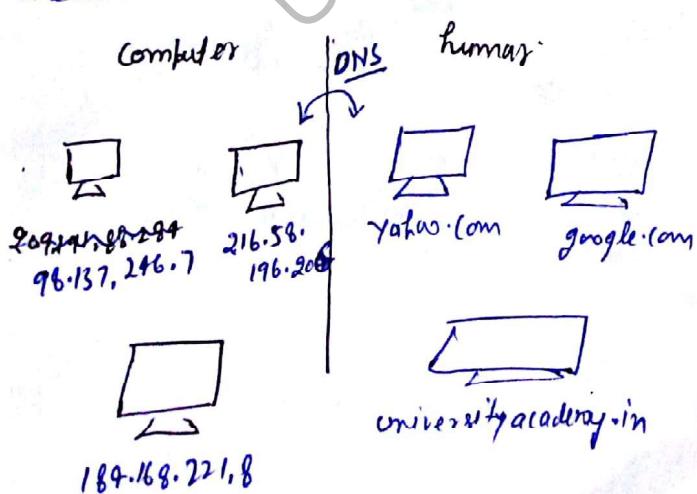
2. Emails are stored on single device

3. Sent email are stored on stored on single devices

4.

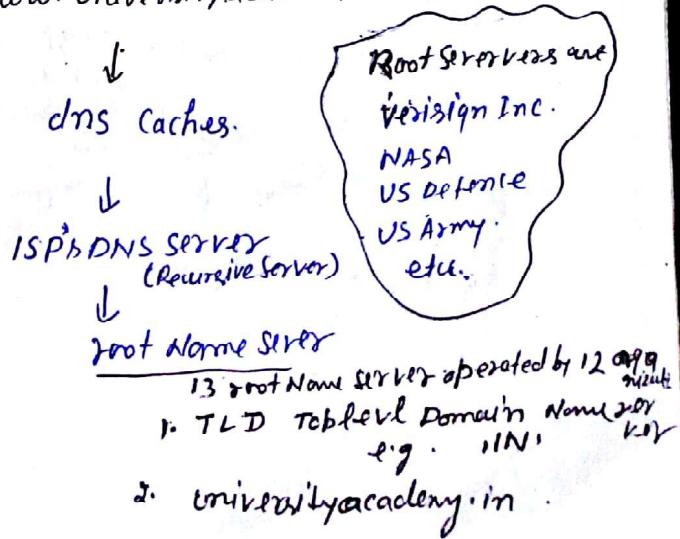
IMAP  
(Internet message Access protocol)  
1. does sync. folder.  
2. Emails are stored on the server  
3. Sent email are stored on server.

(V) DNS: (Domain Name System)  
the DNS is a system that internet domain name are located and translated into IP addresses.



cmd: tracepath www.google.com.  
ping - www.google.com.

www.universityacademy.in → 184.168.221.8

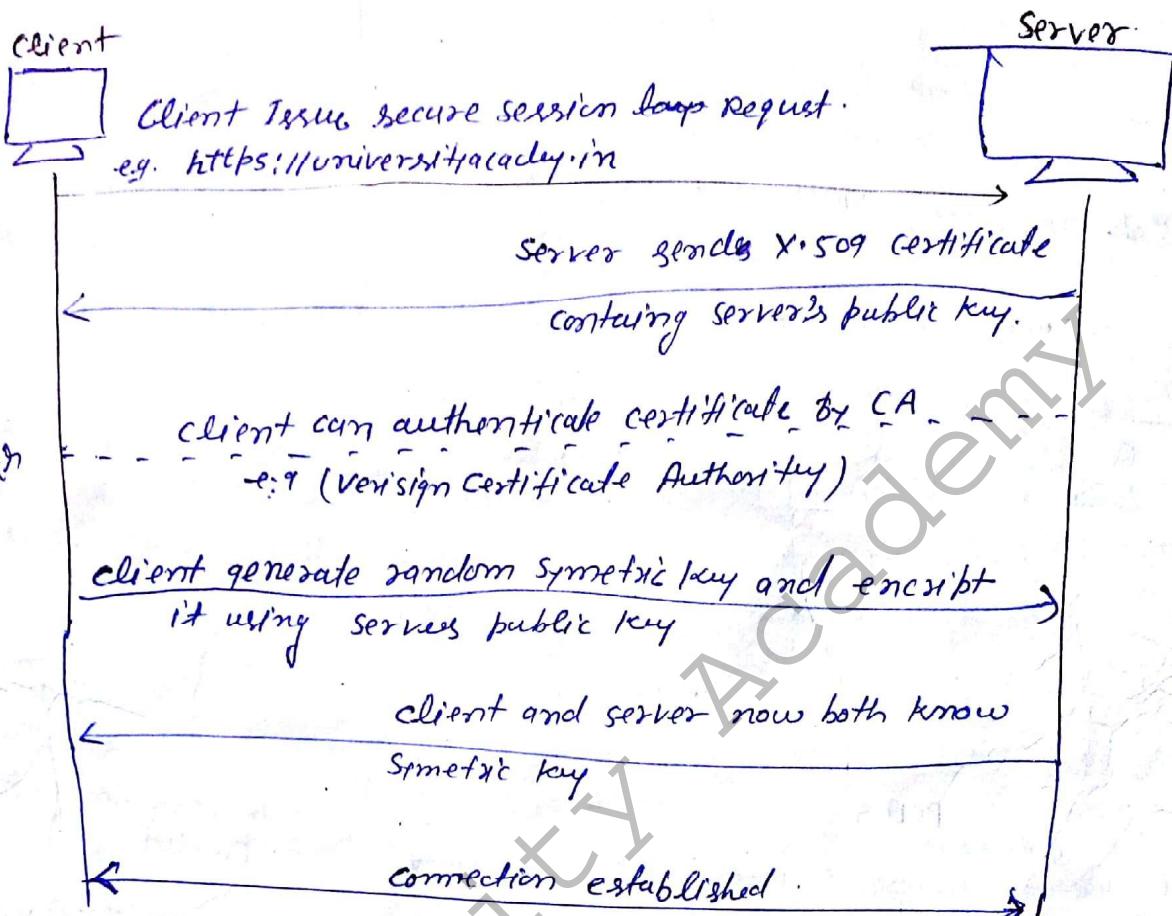


## (VII) SSL

HTTPS = HTTP + SSL

/  
is secure protocol works on top of HTTP  
to provide security.

Indian CA's MC.  
(National Information  
center)



## 2. Transport layer protocol

(i) TCP Transmission Control Protocol. (ii) UDP User Datagram Protocol.  
TCP/IP TCP over IP                                  UDP/IP UDP over IP

a) TCP guarantees the recipient will receive the packets in order by numbering them.

b) UDP does not guarantee for some.

b) Reliable - if packet get lost server will re-request the last part.

b) Unreliable.

c) Connection oriented (follow same path to sent all packet).

c) connection-less.

d) Segmentation (Break into pieces of data)

d) No sequencing.

e) Acknowledge

e) No acknowledgement.

### 3. Network Layer

" IP : An Internet protocol is a numerical label assigned to each device connected to a computer network that uses Internet protocol for communication.

IP Address divided into two major function Part.

1. Host/Network identification

2. Location addressing, host ID.

e.g. 192. 168. 1. 34  
Network ID      host id

An IP address is always set up 4 numbers each ranging from

0 - 255 e.g.: 0. 0. 0. 0 to 255. 255. 255. 255

192. 168. 1. 34  
11000000. 10101000. 00000001. 00100010  
8 bit binary number

Network ID is: 192. 168. 1. 0

Host ID is 34

The IP Address system is divided into 5 classes:

N - Network, H - Host

(1-127)

0NNNNNNN. HHHHHHHH. H-H-H-H-

Class A

Allocating IP Address

(128-191). (0-255)

10NNNNNN. NNNNNNNN. H-H-H-H-

Class B

Allocating IP Address

(192-223). (0-255). (0-255)

110NNNNN. N--N-N--N. H-H-

Class C

Allocating IP Address

(224-239). (0-255). (0-255) (0-255)

First Four bit range from:

Class D

Allocating IP Address

240. 0. 0. 0. to 255. 255. 255. 255.

Class E

Allocating IP Address

## Writing Web Projects

If we writing web project we can use Java technology and tools. A programmer must follows the following steps.

### 1. Web project developing stages.

- Strategy phase : determining the objective of project.
- Design phase : designing the user interface.
- Production phase : building the site.
- Testing phase : testing the site and getting ready to publish on web server.

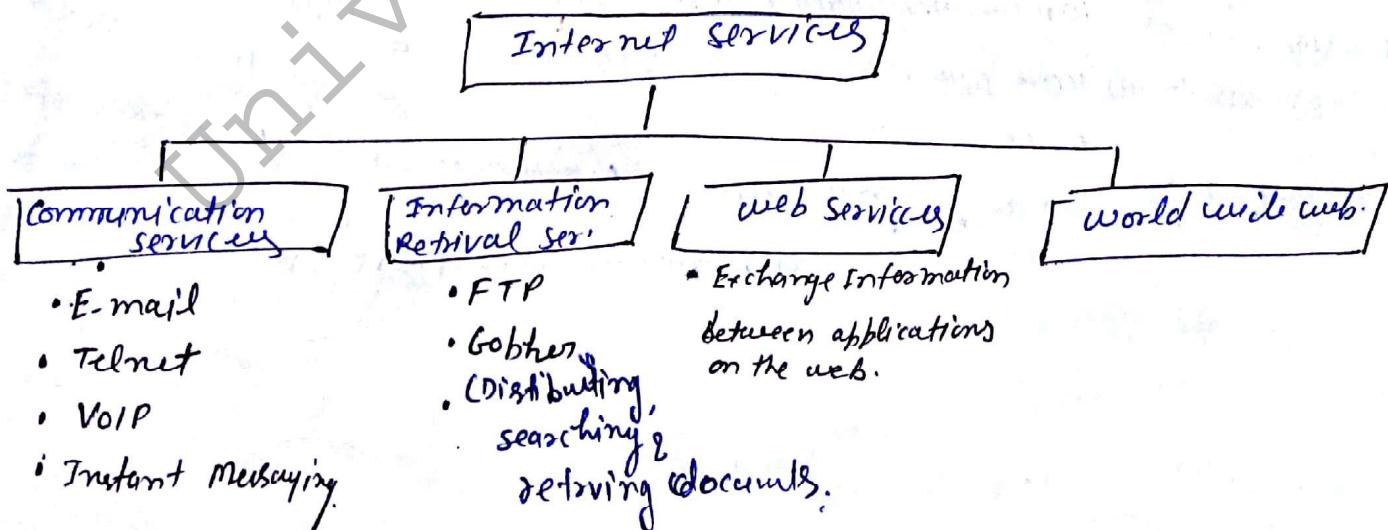
### 2. Web project characteristics.

- Project managers are not always client managers.
- Development schedule.

### 3. Technical specification: is a document that clearly details how a technical component will built.

## Introduction to Internet services and tools:

Internet services allows us to access huge amount of information such as text, graphics, sound and etc over the Internet. There are four categories of Internet services.



Lecture Notes  
On  
**Web Technology**  
  
*By*  
**Mr. Sandeep Vishwakarma**  
*Assistant Professor*  
**Dr. A.P.J. Abdul Kalam**  
*Technical University,Lucknow*

---

**University Academy**  
**Teaching | Training | Informative**

Email: universityacademy.in@gmail.com, Website: www.universityacademy.in,  
YouTube: www.youtube.com/c/UniversityAcademy, Contact: +91-9411988382, +91-989

## **Web Technology**

### **Unit-I**

#### **JAVA**

##### **Introduction:**

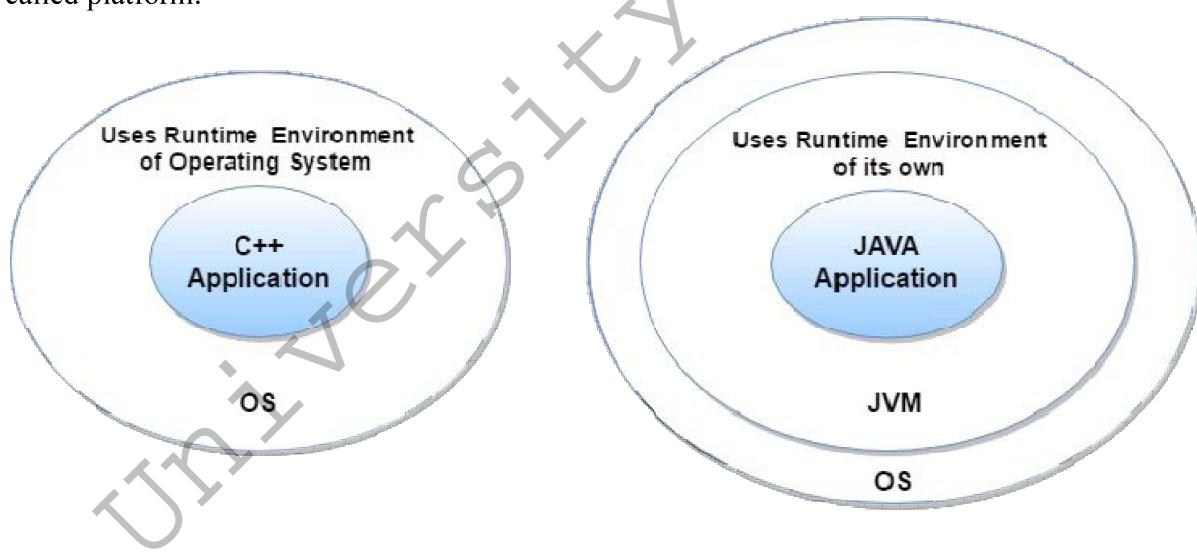
You use word processors to write documents, Web browsers to explore the Internet, and email programs to send email. These are all examples of software that runs on computers. Software is developed using programming languages. There are many programming languages—so *why Java?*

The answer is that Java enables users to develop and deploy applications on the

- Internet for servers,
- desktop computers, and
- small hand-held devices

The future of computing is being profoundly influenced by the Internet, and Java promises to remain a big part of that future. Java is *the Internet programming language*.

Java is a programming language and a platform. Java is a high level, robust, secured and object-oriented programming language. Any hardware or software environment in which a program runs is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.



## History of Java

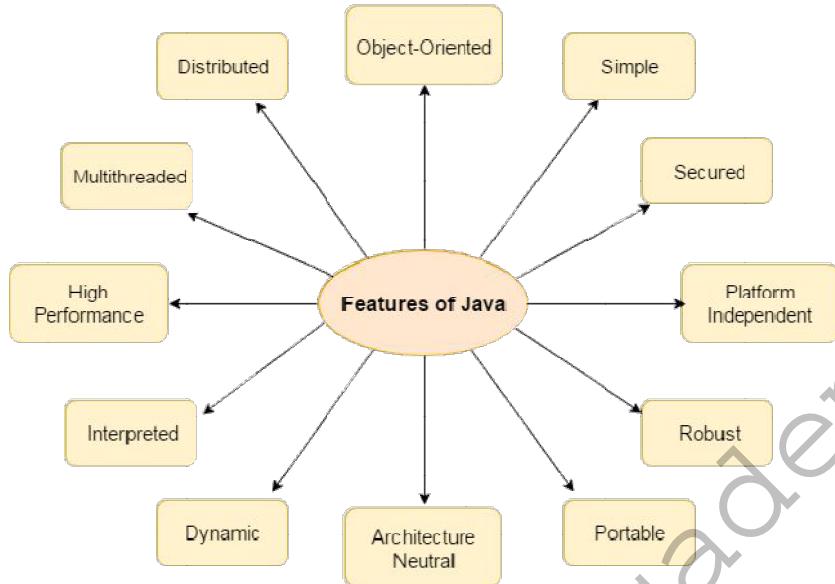
The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts from Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions etc. But, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describe the history of java.

- 1) **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- 2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.
- 3) Firstly, it was called "**Greentalk**" by James Gosling and file extension was .gt.
- 4) After that, it was called **Oak** and was developed as a part of the Green project.
- 5) Why Oak? Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.
- 6) In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.
- 7) Why had they chosen java name for java language? The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.  
According to James Gosling "Java was one of the top choices along with Silk". Since java was so unique, most of the team members preferred java.
- 8) Java is an island of Indonesia where first coffee was produced (called java coffee).
- 9) Notice that Java is just a name not an acronym.
- 10) Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
- 11) In 1995, Time magazine called Java one of the Ten Best Products of 1995.
- 12) JDK 1.0 released in(January 23, 1996).

## Features of Java

The key considerations were summed up by the Java team in the following list of buzzwords:



- Simple:** Java is very easy to learn and its syntax is simple, clean and easy to understand. Java syntax is based on C++. Java has removed many confusing and rarely-used features e.g. explicit pointers, operator overloading etc.
- Object-Oriented:** Java is object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. Basic concepts of OOPs are: Object, Class, Inheritance, Polymorphism, Abstraction and Encapsulation.
- Portable:** Java is platform independent because it is different from other languages like C, C++ etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

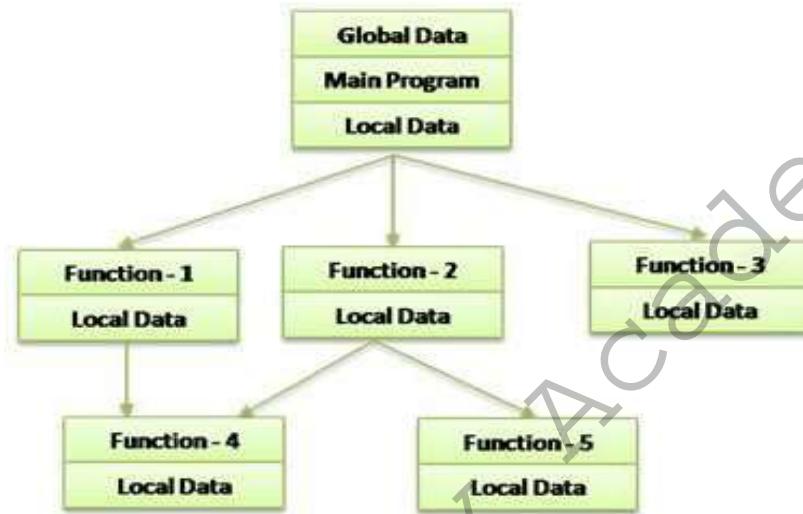
Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA)

- Secured:** Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because: *No explicit pointer, Java Programs run inside virtual machine sandbox*

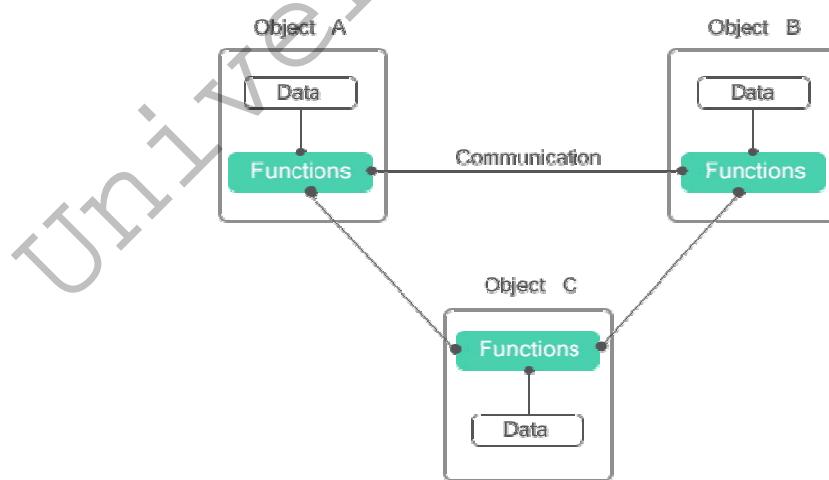
5. **Robust:** Robust simply means strong. Java is robust because: It uses strong memory management. There are lack of pointers that avoids security problems. There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
6. **Architecture neutral:** Java is architecture neutral because there is no implementation dependent features e.g. size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.
7. **Interpreted:** the Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler. Java run-time systems that provide this feature lose none of the benefits of the platform-independent code.
8. **Multithreaded:** A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.
9. **Distributed:** It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages i.e. C and C++.

## Object Oriented Programming:

Before going in detail of OOP we have to understand **procedure oriented programming language**: in the procedural languages the problem is divided in to number of function and each function accomplish a particular task. The primary focus of procedure oriented languages is functions.



**Object Oriented Program:** OOPs treat data as a critical element in th program development and does not allow to flow freely around the system. OOPs allow decomposition of problem in to number of entity called object and build data and method around the object.



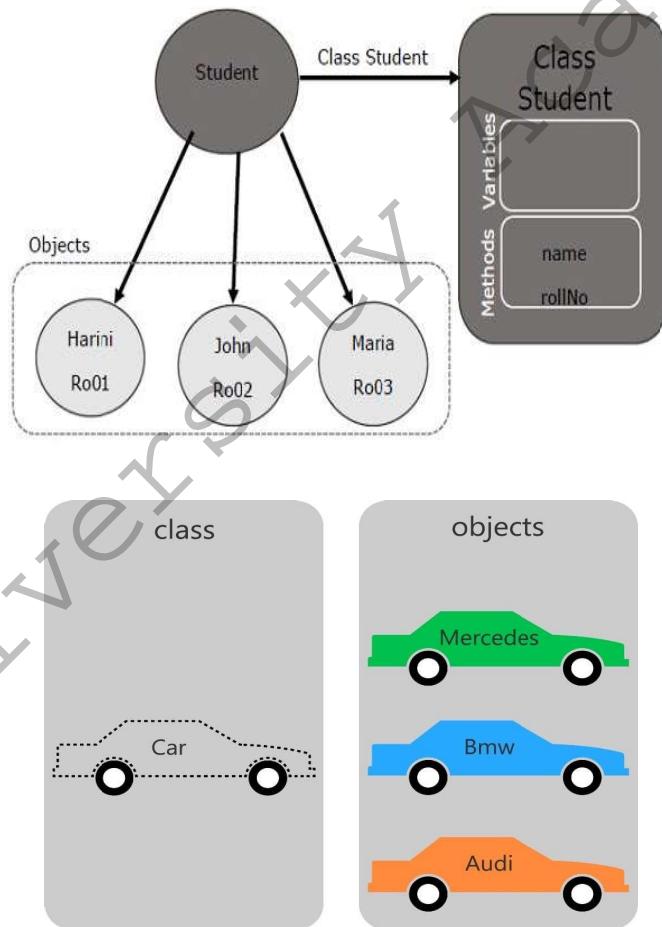
## Basic Concept of OOP:

- Class and Object
- Abstraction
- Encapsulation
- Polymorphism
- Inheritance

### Class and Object

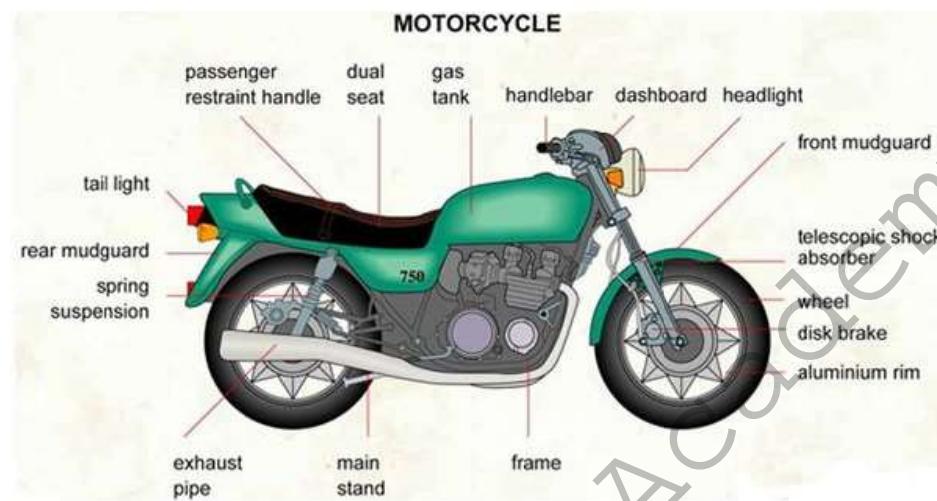
The class is at the core of Java. Class define the shape and nature of object. A class is a *template* for an object, and an object is an *instance* of a class. A class is declared by use of the **class** keyword.

Objects are basic run time entity. It take space in memory. When program is executed the one object interact other object by sending message.



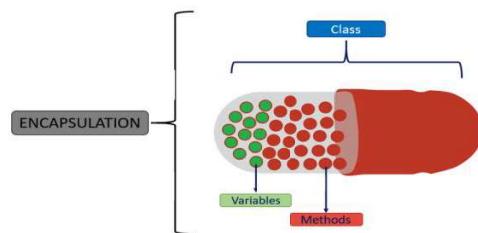
**Abstraction:** Hiding internal detail and showing functionality is known as abstraction. Eg. Phone Call. In java we can use Abstract Class and Interface to achieve abstraction

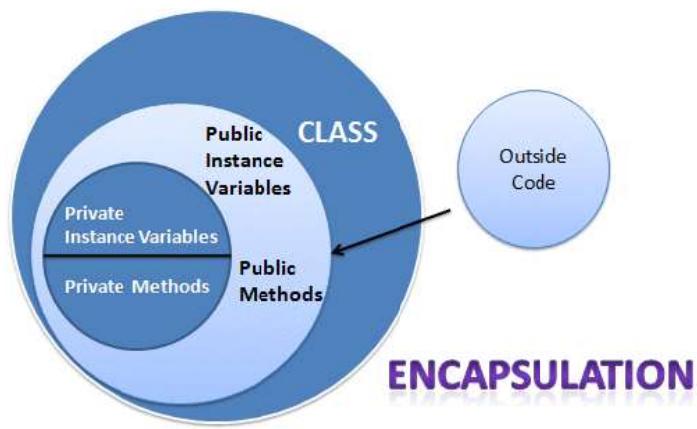
## Abstraction



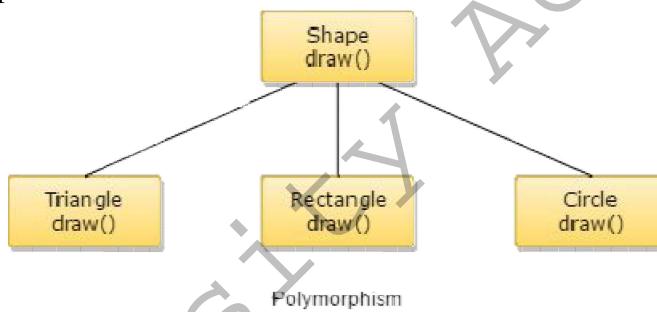
Real Life Example of Abstraction

**Encapsulation:** Binding or wrapping code and data together into single unit is known as encapsulation eg. Capsule

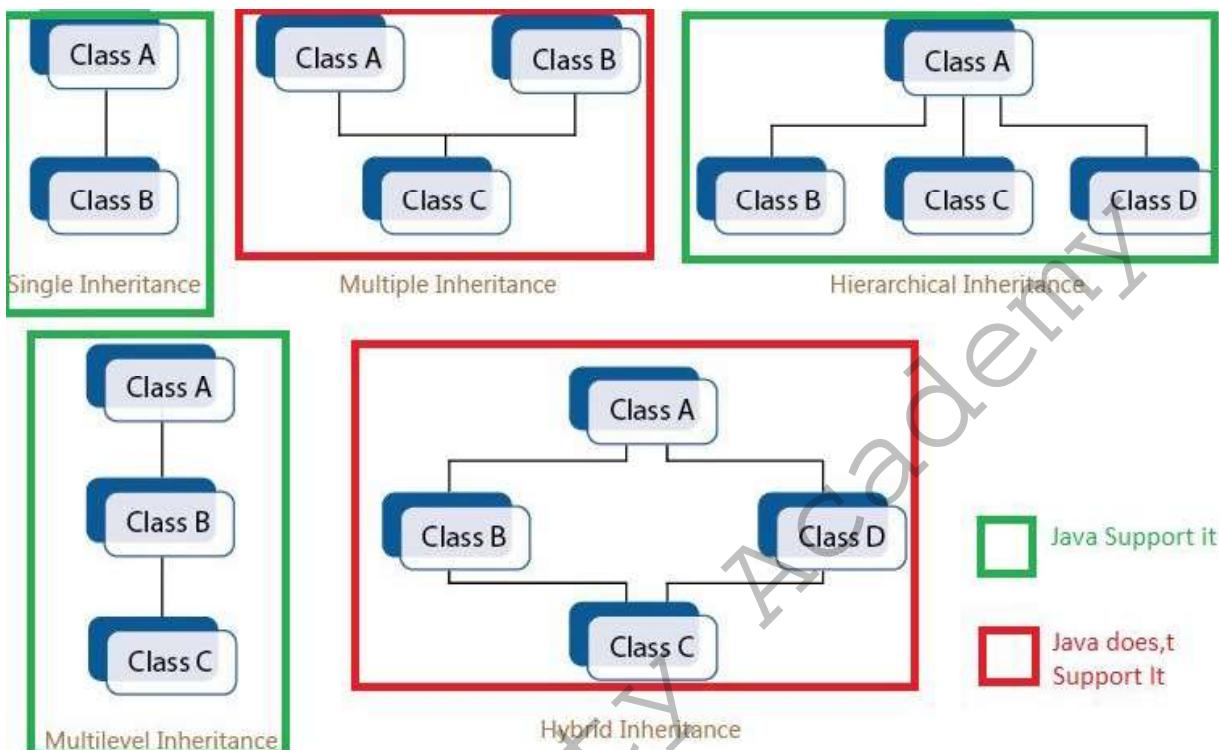




**Polymorphism:** Polymorphism in java is a concept by which we can perform a single action by different ways. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms. Method overloading and Method overriding achieved Polymorphism.



**Inheritance:** The process by which object of one class acquires the properties and functionalities of another parent object of class is called inheritance. It provides code reusability.



## A First Simple Program

For executing any java program, you need to

1. Install the JDK if you don't have installed it, download the JDK and install it.
2. Set path of the jdk/bin directory.
3. create the java program
4. compile and run the java program

```
/*
This is a simple Java program.
Call this file "Hello.java".
*/
Comment class Hello{
    // Your program begins with a call to main().
    public static void main(String args[]) {
        System.out.println("Hello Java.");
    }
}
```

```
C:\Users\Sandeep>cd..
C:\Users>cd..
C:\>cd java
C:\Java>javac Hello.java
C:\Java>java Hello
Hello Java.

C:\Java>
```

### Closer Look at the First Sample Program

**Comment:** Multiline, Single Line, Documentation

**public:** keyword is an access specifier, which allows the programmer to control the visibility of class members

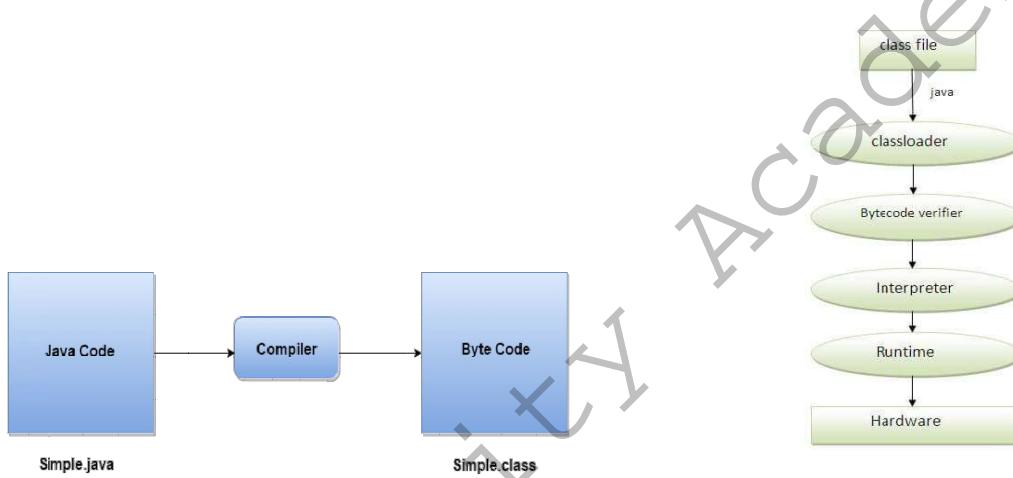
**static:** The keyword static allows main( ) to be called without having to instantiate a particular instance of the class. This is necessary since main( ) is called by the Java Virtual Machine before any objects are made

**void:** The keyword void simply tells the compiler that main( ) does not return a value.

**main:** main( ) is the method called when a Java application begins. Keep in mind that Java is case-sensitive. Thus, Main is different from main. It

**String args[ ]:** In main( ), there is only one parameter, albeit a complicated one. String args[ ] declares a parameter named args, which is an array of instances of the class String. (Arrays are collections of similar objects.) Objects of type String store character strings. In this case, args receives any command-line arguments present when the program is executed.

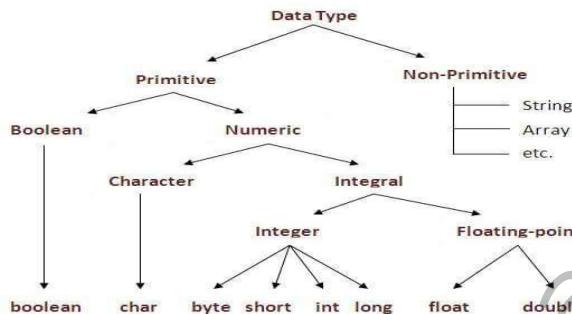
**System.out.println:** System is a predefined class that provides access to the system, and out is the output stream that is connected to the console. println( ) displays the string which is passed to it.



## Data Types in Java

Primitive data types: The primitive data types include Integer, Floating Point, Character and Boolean.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.



Data Type	Default Value	Default size	Range	Used As
boolean	false	1 bit	true or false	boolean b;
char	'\u0000'	2 byte	0 to 65,536	char ch1, ch2;
byte	0	1 byte	-128 to 127	byte b, c;
short	0	2 byte	-32,768 to 32,767	short t;
int	0	4 byte	-2,147,483,648 to 2,147,483,647	int a;
long	0L	8 byte	- 9,223,372,036,854,775, 808 to 9,223,372,036,854,775, 807	long days;
float	0.0f	4 byte	1.4e-045 to 3.4e+038	float avg, sum;
double	0.0d	8 byte	4.9e-324 to 1.8e+308	double pi, r, a;

## Java Variables

Variable is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed. A variable is defined by the combination of an identifier, a type, and an optional initializer. all variables have a scope, which defines their visibility, and a lifetime.

```
Data_type identifier [= value][, variable [= value] ...];
```

Examples:

```
int a, b, c;           // Declares three ints, a, b, and c.  
int a = 10, b = 10;   // Example of initialization  
byte B = 22;          // initializes a byte type variable B.  
double pi = 3.14159; // declares and assigns a value of PI.  
char a = 'a';          // the char variable a is initialized with value 'a'
```

There are three types of variables in java:

- o local variable: A variable declared inside the body of the method
- o instance variable: A variable declared inside the class but outside the body of the method
- o static variable: A variable which is declared as static is called static variable. It cannot be local

```
class A{  
    int data=50;//instance variable  
    static int m=100;//static variable  
    void method(){  
        int n=90;//local variable  
    }  
}
```

## Identifiers in Java

All Java components require names. Name used for classes, methods, interfaces and variables are called **Identifier**. Identifier must follow some rules. Here are the rules:

- All identifiers must start with either a letter( a to z or A to Z ) or currency character(\$) or an underscore.
- After the first character, an identifier can have any combination of characters.
- A Java **keyword** cannot be used as an identifier.
- Identifiers cannot be true, false or null.
- Identifiers in Java are case sensitive; foo and Foo are two different identifiers.

Samples of acceptable variable names: YES	Samples of unacceptable variable names: NO
Grade	Grade(Test)
GradeOnTest	GradeTest#1
Grade_On_Test	3rd_Test_Grade
GradeTest	Grade Test (has a space)

## Operators in java

**Operator** in java is a symbol that is used to perform operations. For example: +, -, \*, / etc.

There are many types of operators in java which are given below:

- o Unary Operator,
- o Arithmetic Operator,
- o Relational Operator,
- o Bitwise Operator,
- o Logical Operator,
- o Assignment Operator.

### The Unary Operators

The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.

Operator	Description
+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

### Arithmetic Operators

The Java programming language provides operators that perform addition, subtraction, multiplication, and division. There's a good chance you'll recognize them by their counterparts in basic mathematics. The only symbol that might look new to you is "%", which divides one operand by another and returns the remainder as its result.

Operator	Description
+	Additive operator (also used for String concatenation)
-	Subtraction operator

*	Multiplication operator
/	Division operator
%	Remainder operator

## Relational Operators

The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand. The majority of these operators will probably look familiar to you as well. Keep in mind that you must use "==" , not "=". when testing if two primitive values are equal.

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

## Logical Operator

&	Conditional-AND
	Conditional-OR
?:	Ternary (shorthand for if-then-else statement)

## Assignment Operator

=	Simple assignment operator
---	----------------------------

## Bitwise Operators

The bitwise & operator performs a bitwise AND operation.

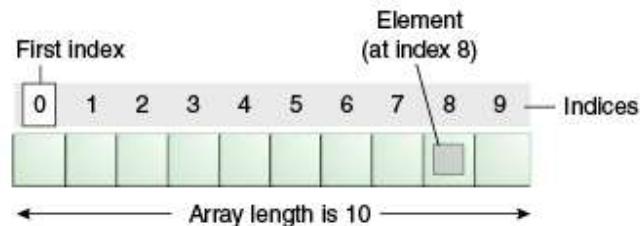
The bitwise ^ operator performs a bitwise exclusive OR operation.

The bitwise | operator performs a bitwise inclusive OR operation

shift operator "<<" shifts a bit pattern to the left, and the signed right shift operator ">>" shifts a bit pattern to the right.

## Arrays

An *array* is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. Array in java is index based, first element of the array is stored at 0 index.



### I. One-Dimensional Arrays

To create an array, you first must create an array variable of the desired type. The general form of a one-dimensional array declaration is

<i>type array-var[];</i>	- declaration
<i>array-var = new type[size];</i>	- instantiation

*array-var* is the array variable that is linked to the array  
The elements in the array allocated by **new** will automatically be

initialized to zero

Example:

```
int month_days[];
month_days = new int[12];
```

Or

```
int month_days[] = new int[12];
```

Or

```
int a[]={33,3,4,5};
```

*Example1:*

```
class Array {
    public static void main(String args[]) {
        int month_days[]; //Declaration
        month_days = new int[12]; // instantiation
        month_days[0] = 31; // initialization
        month_days[1] = 28;
        month_days[2] = 31;
        month_days[3] = 30;
        month_days[4] = 31;
        month_days[5] = 30;
        month_days[6] = 31;
        month_days[7] = 31;
        month_days[8] = 30;
        month_days[9] = 31;
        month_days[10] = 30;
        month_days[11] = 31;
```

```

        System.out.println("April has " + month_days[3] + " days.");
    }
}

```

Output:

```

C:\Java>javac Array.java
C:\Java>java Array
April has 30 days.

```

*Example2:*

```

class Testarray1{
public static void main(String args[]){
    int a[]={33,3,4,5}//declaration, instantiation and initialization
    //printing array
    for(int i=0;i<a.length;i++)//length is the property of array
    System.out.println(a[i]);
}

```

Output

```

C:\Java>javac Testarray1.java
C:\Java>java Testarray1
33
3
4
5

```

## II. Multidimensional Arrays

Data is stored in row and column based index (also known as matrix form). *multidimensional arrays* are actually arrays of arrays

```
int arr[][] =new int[3][4];//3 row and 4 column
```

Example:

```

// Demonstrate a two-dimensional array.
class TwoDArray {
public static void main(String args[]) {
    int twoD[][]= new int[4][5];
    int i, j, k = 0;
    for(i=0; i<4; i++)
        for(j=0; j<5; j++) {
            twoD[i][j] = k;
            k++;
        }
}

```

```

for(i=0; i<4; i++) {
    for(j=0; j<5; j++)
        System.out.print(twoD[i][j] + " ");
    System.out.println();
}
}
}
}

```

**Output:**

```

C:\Java>javac TwoDArray.java

C:\Java>java TwoDArray
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19

```

**Classes And Method**

The class is at the core of Java. It is the logical construct upon which the entire Java language is built because it defines an object. The class forms the basis for object-oriented programming in Java. Thus, a class is a *template* for an object, and an object is an *instance* of a class. A class is declared by use of the **class** keyword. A simplified general form of a class definition is shown here

```

class classname {
    type instance-variable1;
    type instance-variable2;
    // ...
    type instance-variableN;
    type methodname1(parameter-list) {
        // body of method
    }
    type methodname2(parameter-list) {
        // body of method
    }
    // ...
    type methodnameN(parameter-list) {
        // body of method
    }
}

```

The methods and variables defined within a class are called *members* of the class. All methods have the same general form as **main( )**, most methods will not be specified as **static** or **public**. Java classes do not need to have a **main( )** method. You only specify one if that class is the starting point for your program. Further, applets don't require a **main( )** method at all.

**Example:**

```

class Box {
    double width;
}

```

```
        double height;
        double depth;
    }
```

a class defines a new type of data the new data type is called **Box**. You will use this name to declare **objects** of type **Box**. you will use a statement like the following:

```
Box mybox = new Box(); // create a Box object called mybox
```

Every Box object will contain its own copies of the instance variables width, height, and depth. To access these variables, you will use the dot (.) operator.

```
mybox.width = 100;
```

Program:

```
/* A program that uses the Box class.
Call this file BoxDemo.java
*/
class Box {
    double width;
    double height;
    double depth;
}
// This class declares an object of type Box.
class BoxDemo {
    public static void main(String args[]) {
        Box mybox = new Box();
        double vol;
        // assign values to mybox's instance variables
        mybox.width = 10;
        mybox.height = 20;
        mybox.depth = 15;
        // compute volume of box
        vol = mybox.width * mybox.height * mybox.depth;
        System.out.println("Volume is " + vol);
    }
}
```

Output

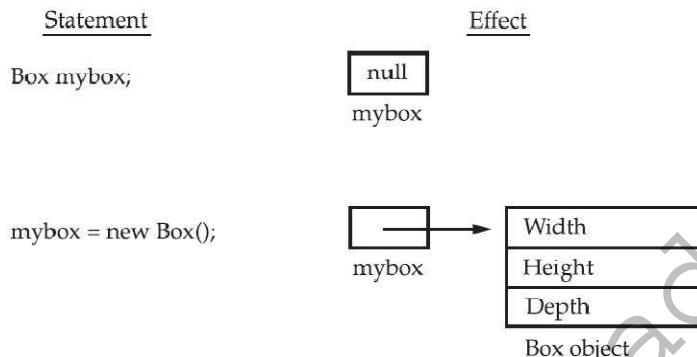
```
C:\Java>javac BoxDemo.java
C:\Java>java BoxDemo
Volume is 3000.0
```

**Declaration of Object:** The **new** operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it.

```
Box mybox = new Box(); //declare reference to object and allocate a Box object
```

Or

```
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```



### **Program:**

```
// This program uses a parameterized method.
class Box {
    double width;
    double height;
    double depth;
    // compute and return volume
    double volume() {
        return width * height * depth;
    }
    // sets dimensions of box
    void setDim(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }
}
class BoxDemo5 {
    public static void main(String args[]) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;
        // initialize each box
        mybox1.setDim(10, 20, 15);
        mybox2.setDim(3, 6, 9);
    }
}
```

```
// get volume of first box  
vol = mybox1.volume();  
System.out.println("Volume is " + vol);  
// get volume of second box  
vol = mybox2.volume();  
System.out.println("Volume is " + vol);  
}  
}
```

## Output

```
C:\Java>javac BoxDemo5.java  
  
C:\Java>java BoxDemo5  
Volume is 3000.0  
Volume is 162.0
```

## Introducing Methods:

lasses usually consist of two things: instance variables and methods

This is the general form of a method:

```
type name(parameter-list) {  
    // body of method  
}
```

**type** specifies the type of data returned by the method

The **name** of the method is specified by name

The **parameter-list** is a sequence of type and identifier pairs separated by commas

Methods that have a return type other than void return a value to the calling routine using the following form of the return statement:

```
return value;
```

Example:

```
class Box {  
    double width;  
    double height;  
    double depth;  
    // display volume of a box  
    void volume() {  
        System.out.print("Volume is ");  
        System.out.println(width * height * depth);  
    }  
}
```

```
}

class BoxDemo3
{
    public static void main(String args[])
    {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        // assign values to mybox1's instance variables
        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;
        /* assign different values to mybox2's
        instance variables */
        mybox2.width = 3;
        mybox2.height = 6;
        mybox2.depth = 9;
        // display volume of first box
        mybox1.volume();
        // display volume of second box
        mybox2.volume();
    }
}
```

**Method That Takes Parameters**

```
class Box {
    double width;
    double height;
    double depth;
    // compute and return volume
    double volume() {
        return width * height * depth;
    }
    // sets dimensions of box
    void setDim(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }
}

class BoxDemo5 {
    public static void main(String args[])
    {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;
        // initialize each box
        mybox1.setDim(10, 20, 15);
        mybox2.setDim(3, 6, 9);
        // get volume of first box
        vol = mybox1.volume();
    }
}
```

```
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

## Inheritance in Java

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

### Terms used in Inheritance

- o **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- o **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- o **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- o **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

### The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class

- **Single Inheritance Example**

```
class Animal{  
void eat()  
{  
System.out.println("eating...");}  
}  
class Dog extends Animal{  
void bark()  
{  
System.out.println("barking...");}  
}  
class TestInheritance{  
public static void main(String args[]){  
Dog d=new Dog();  
d.bark();  
d.eat();  
}  
}
```

- **Multilevel Inheritance Example**

```
class Animal{  
void eat(){System.out.println("eating...");}  
}  
class Dog extends Animal{  
void bark(){System.out.println("barking...");}  
}  
class BabyDog extends Dog{  
void weep(){System.out.println("weeping...");}  
}  
class TestInheritance2{  
public static void main(String args[]){  
BabyDog d=new BabyDog();  
d.weep();  
d.bark();  
d.eat();  
}  
}
```

- **Hierarchical Inheritance Example**

```
class Animal{  
    void eat(){System.out.println("eating...");}  
}  
class Dog extends Animal{  
    void bark(){System.out.println("barking...");}  
}  
class Cat extends Animal{  
    void meow(){System.out.println("meowing...");}  
}  
class TestInheritance3{  
    public static void main(String args[]){  
        Cat c=new Cat();  
        c.meow();  
        c.eat();  
        //c.bark();//C.T.Error  
    }  
}
```



**University Academy**

Teaching|Training|Informative

# **Web Technology**

## **(Java programming)**

**Unit-I**

### ***Java Applet***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Introduction**
- **Advantage**
- **Lifecycle of Java Applet**
- **Example**
- **Importing Packages**
- **Access Specifiers**

# Introduction

- Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.
- applet does not have a **main( ) method**
- An **applet** is a Java program that runs in a Java-compatible Web browser or standard tool, **appletviewer**.
- This applet begins with two **import statements**
  - import java.awt.\*; // Abstract Window Toolkit
  - import java.applet.\*;

# Advantage

- There are many advantages of applet. They are as follows:
  - It works at client side so less response time.
  - Secured
  - It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.
- Drawback of Applet
  - Plugin is required at client browser to execute applet.

# Lifecycle of Java Applet

## **java.applet.Applet**

- **Applet is initialized:** **public void init()** is used to initialize the Applet. It is invoked only once
- **Applet is started:** **public void start()** is invoked after the init() method or browser is maximized. It is used to start the Applet.
- **Applet is stopped:** **public void stop()**: is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- **Applet is destroyed:** **public void destroy()**: is used to destroy the Applet. It is invoked only once.

## **java.awt.Component**

- **Applet is painted:** **public void paint(Graphics g)** invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser

# Applet Example

- Simple example of Applet by html file:

```
//First.java  
import java.applet.Applet;  
import java.awt.Graphics;  
public class First extends Applet{  
  
public void paint(Graphics g){  
g.drawString("welcome",150,150);  
}  
}
```

```
<html>  
<body>  
<applet code="First.class" width="300" hei  
ght="300">  
</applet>  
</body>  
</html>
```

- To execute the applet by html file, create an applet and compile it.
- After that create an html file and place the applet code in html file

```
c:\>javac First.java  
c:\>appletviewer myapplet.html
```

# Applet Example

- Simple example of Applet by appletviewer tool:

```
//Second.java  
import java.applet.Applet;  
import java.awt.Graphics;  
public class Second extends Applet{  
    public void paint(Graphics g){  
        g.drawString("welcome to applet",150,150);  
    }  
}  
/*  
<applet code="Second.class" width="300" height="300">  
</applet>  
*/
```

```
c:\>javac Second.java  
c:\>appletviewer Second.java
```



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-I**

***AWT(Abstract Window Toolkit)***

***in Java***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

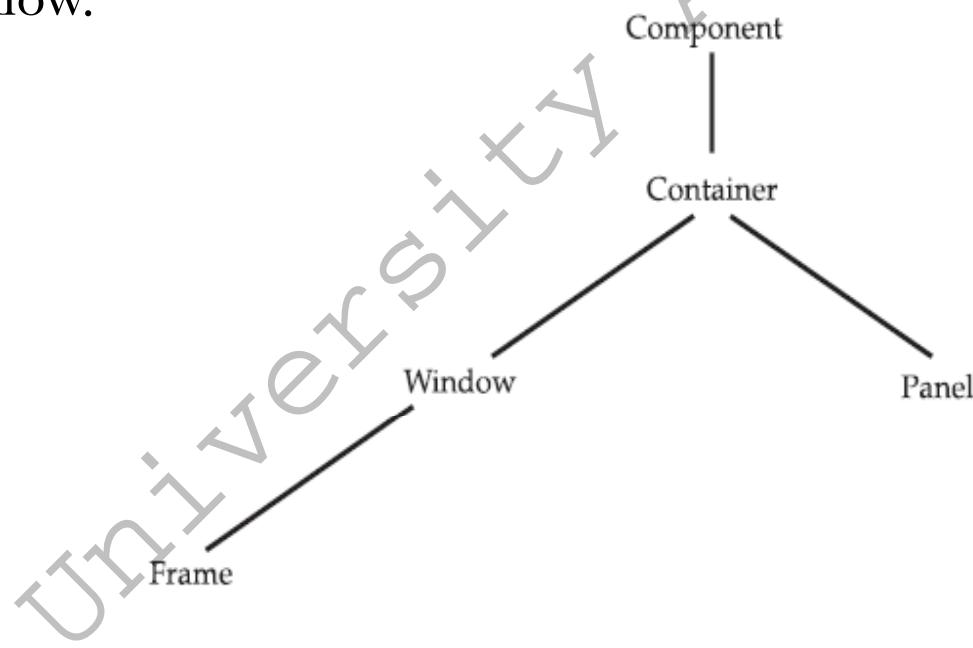
- **Introduction to AWT**
- **AWT Control**
- **AWT Layout Managers**
- **Examples**

# Introduction

- The Graphics programming in java is supported by AWT package.
- The AWT contains large number of classes which help to include various graphical component in the java program.
- These components are text box,button,labels, radio button, list item and so on.
- We need to import **java.awt** package. It is one of Java's largest packages.
- These classes are arranged in hierarchical manner.

# Window Fundamentals

- The two most common windows are those
  - derived from **Panel**, which is used by applets,
  - derived from **Frame**, which creates a standard application window.



# Window Fundamentals

- **Component:** It defines over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting. A **Component object** is responsible for remembering the current foreground and background colors
- **Container:** The Container class is a subclass of Component. It has additional methods that allow other Component objects to be nested within it. Other Container objects can be stored inside of a Container. A container is responsible for layout and placement of graphical component.
- **Window:** the top level window without border and without menu bar is created using **window class**.
- **Panel:** The Panel class is a subclass of Container. It similar to Window without border and without any menu bar , title bar.
- **Frame:** it is top level window with border and menu bar. It support the common window event such as window open, close.

# AWT Control

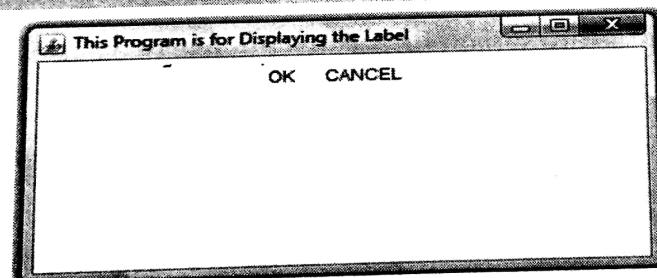
- Label
- Button
- Text Box
- Check Box
- Choice

## Java Program[Use\_Label.java]

```
import java.awt.*;
class Use_Label
{
    public static void main(String[] args)
    {
        Frame fr=new Frame("This Program is for Displaying the Label");
        fr.setSize(400,200);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Label L1=new Label("OK");
        Label L2=new Label("CANCEL");
        fr.add(L1);
        fr.add(L2);
    }
}
```

## Output

```
C:\>javac Use_label.java
C:\>java Use_label
```



# Button

- Button

Button (String s)

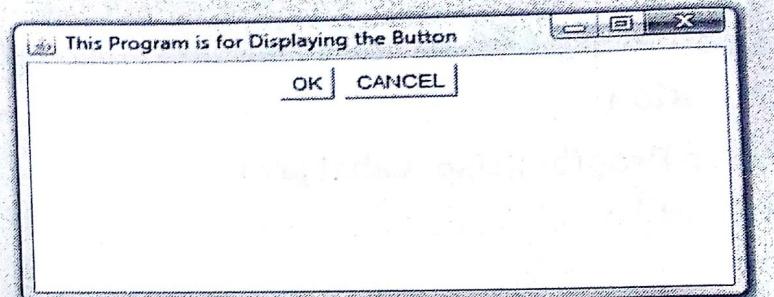
## Java Program

```
import java.awt.*;
class Use_Button
{
    public static void main(String[] args)
    {
```

```
        Frame fr=new Frame("This Program is for Displaying the Button");
        fr.setSize(400,200);
```

```
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Button B1 = new Button("OK");
        Button B2 = new Button("CANCEL");
        fr.add(B1);
        fr.add(B2);
    }
}
```

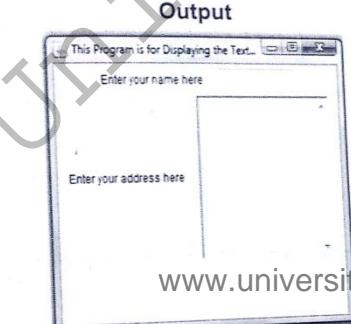
## Output



# Text Box

Java Program[Use\_TxtFld.java]

```
import java.awt.*;
class Use_TxtFld
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the TextField");
        fr.setSize(350,300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Label L1=new Label("Enter your name here");
        TextField input1=new TextField(10);
        Label L2=new Label("Enter your address here");
        TextArea input2=new TextArea(10,20);
        fr.add(L1);
        fr.add(input1);
        fr.add(L2);
        fr.add(input2);
    }
}
```



# Check Box

**Checkbox(String label)**

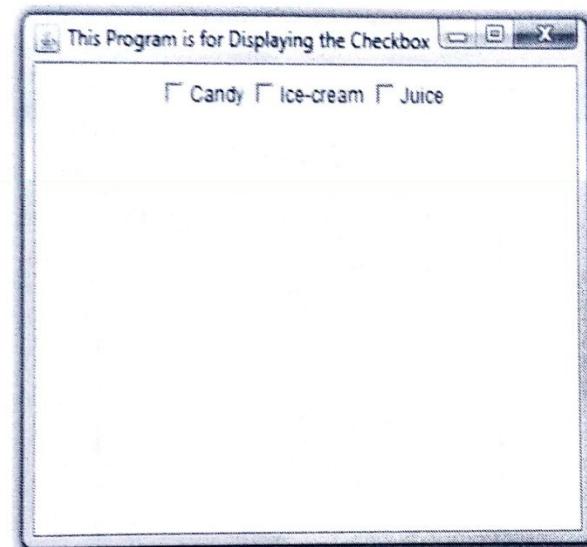
where *label* denotes the label associated with each checkbox.

To get the state of the checkbox the **getState()** method can be used.

**Java Program[Use\_ChkBox.java]**

```
import java.awt.*;
class Use_ChkBox
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the Checkbox");
        fr.setSize(350,300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Checkbox box1=new Checkbox("Candy");
        Checkbox box2=new Checkbox("Ice-cream");
        Checkbox box3=new Checkbox("Juice");
        fr.add(box1);
        fr.add(box2);
        fr.add(box3);
    }
}
```

**Output**



# Choice

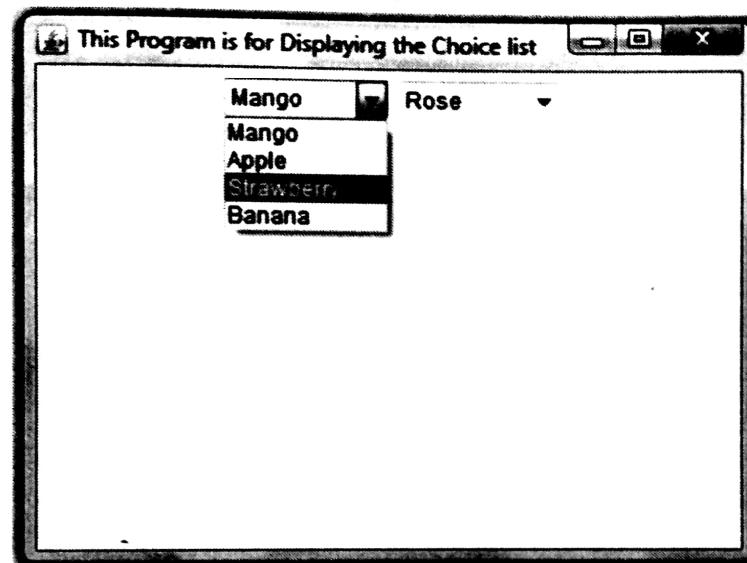
```
Choice obj=new Choice();
```

## Java Program[Use\_CheckBox.java]

```
import java.awt.*;
class Use_CheckBox
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the Choice list");
        fr.setSize(350,300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Choice c1=new Choice();
        Choice c2=new Choice();
        c1.add("Mango");
        c1.add("Apple");
        c1.add("Strawberry");
        c1.add("Banana");

        c2.add("Rose");
        c2.add("Lily");
        c2.add("Lotus");
        fr.add(c1);
        fr.add(c2);
    }
}
```

## Output



# AWT Layout Managers

- **FlowLayout:** FlowLayout is the default layout manager. FlowLayout implements a simple layout style, which is similar to how words flow in a text editor. The direction of the layout is governed by the container's component orientation property, which, by default, is left to right, top to bottom
- **BorderLayout:** The BorderLayout class implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area in the center. The four sides are referred to as north south, east, and west. The middle area is called the center.



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-I**

## ***Event Handling in Java***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Introduction**
- **Components of Event Handling**
- **Types of Event Handling**
- **Examples**

# Introduction

- Event describes the change in state of any object. It interrupt the current ongoing activity. Eg. When user clicks mouse or press keyboard during some processing, then it generate an activity.
- Event represent all activity that carried out between user and application.
- Java abstract window toolkit(AWT) conveys these message to the program.
- Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

# Components of Event Handling

- Event handling has three main components,
  - **Events** : An event is a change in state of an object.
  - **Events Source** : Event source is an object that generates an event.
  - **Listeners** : A listener is an object that listens to the event. A listener gets notified when an event occurs.

# Types of Event Handling

- There are two commonly used events :
  - **Mouse Event** : While handling the mouse event we use two interface **MouseListener** and **MouseMotionListener**. These interface has certain methods such as *mouseClicked()*, *mousePressed()*, *mouseReleased()*, *mouseEntered()*, *mouseExited()*, *mouseDragged()* and *mouseMoved()*.
  - **Keyboard Event**: When Key from keyboard is pressed then it caused an event. There are three commonly method from **keyListener** interface and those are *keyPressed()*, *keyReleased()* and *keyTyped()*.

```

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents" width=300
height=300>
</applet>
*/
public class MouseEvents extends Applet
implements MouseListener,
MouseMotionListener {
String msg = "";
int mouseX = 0, mouseY = 0; // coordinates
of mouse
public void init() {
addMouseListener(this);
addMouseMotionListener(this);
}
// Handle mouse clicked.
public void mouseClicked(MouseEvent me) {
// save coordinates
mouseX = 0;
mouseY = 10;
msg = "Mouse clicked.";
repaint();
}

```

```

// Handle mouse entered.
public void mouseEntered(MouseEvent me) {
// save coordinates
mouseX = 0;
mouseY = 10;
msg = "Mouse entered.";
repaint();
}
// Handle mouse exited.
public void mouseExited(MouseEvent me) {
// save coordinates
mouseX = 0;
mouseY = 10;
msg = "Mouse exited.";
repaint();
}
// Handle button pressed.
public void mousePressed(MouseEvent me) {
// save coordinates
mouseX = me.getX();
mouseY = me.getY();
msg = "Down";
repaint();
}
// Handle button released.
public void mouseReleased(MouseEvent me) {
// save coordinates
mouseX = me.getX();
mouseY = me.getY();
msg = "Up";
repaint();
}
// Handle mouse dragged.
public void
mouseDragged(MouseEvent me) {
// save coordinates
mouseX = me.getX();
mouseY = me.getY();
msg = "*";
showStatus("Dragging mouse at " +
mouseX + ", " + mouseY);
repaint();
}
// Handle mouse moved.
public void mouseMoved(MouseEvent
me) {
// show status
showStatus("Moving mouse at " +
me.getX() + ", " + me.getY());
}
// Display msg in applet window at
current X,Y location.
public void paint(Graphics g) {
g.drawString(msg, mouseX, mouseY);
}
}

```

```

// Demonstrate the key event handlers.

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="SimpleKey" width=300
height=100>
</applet>
*/
public class SimpleKey extends Applet
implements KeyListener {
String msg = "";
int X = 10, Y = 20; // output coordinates
public void init() {
addKeyListener(this);
}

```

```

public void keyPressed(KeyEvent ke) {
showStatus("Key Down");
}
public void keyReleased(KeyEvent ke) {
showStatus("Key Up");
}
public void keyTyped(KeyEvent ke) {
msg += ke.getKeyChar();
repaint();
}
// Display keystrokes.
public void paint(Graphics g) {
g.drawString(msg, X, Y);
}
}

```



**University Academy**

Teaching|Training|Informative

# **Web Technology**

## **(Java programming)**

**Unit-I**

### ***Java Exception Handling***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Introduction**
- **Exception-Handling**
- **Syntax**
- **Exception classes**
- **Types of Java Exceptions**
- **Examples**

# Introduction

- Exception is an unusual condition that can occur in the program
- This condition is cause due to run time error in the program.
- Exception handling mechanism allow the programmer to handle run time error situation gracefully without crashing the program.
- Eg divide by zero, array access out of bound etc.

```
Statement 1;  
Statement 2;  
Statement 3;  
Statement 4;  
Statement 5;//Exception Err0r  
Statement 6;  
Statement 7;  
Statement 8;
```

# Exception-Handling

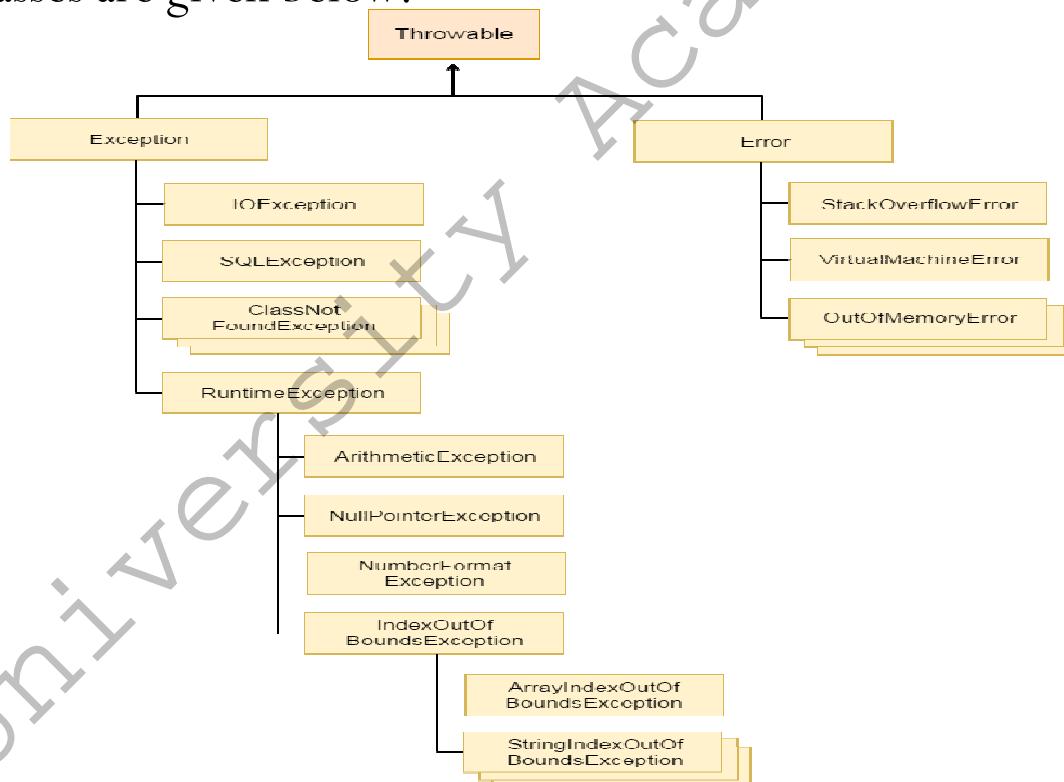
- Java exception handling is managed via five keywords: **try, catch, throw, throws, and finally**.
- **Try** :The "try" keyword is used to specify a block where we should place exception code.
- **Catch**: The "catch" block is used to handle the exception. It must be preceded by try block
- **Finally**: The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
- **Throw** :The "throw" keyword is used to throw an exception.
- **Throws**: The "throws" keyword is used to declare exceptions. It doesn't throw an exception.

# Syntax

```
try {  
    // block of code to monitor for errors  
}  
catch (ExceptionType1 exOb) {  
    // exception handler for ExceptionType1  
}  
catch (ExceptionType2 exOb) {  
    // exception handler for ExceptionType2  
}  
// ...  
finally {  
    // block of code to be executed after try block ends  
}
```

# A hierarchy of Java Exception classes

- The `java.lang.Throwable` class is the root class of Java Exception hierarchy which is inherited by two subclasses: `Exception` and `Error`. A hierarchy of Java Exception classes are given below:



# Types of Java Exceptions

- There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:
- Checked Exception: The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.
- Unchecked Exception: The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
- Error: Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

# Examples

```
public class JavaExceptionExample{  
    public static void main(String args[]){  
        try{  
            //code that may raise exception  
            int data=100/0;  
        }catch(ArithmaticException e){System.out.println(e);}  
        //rest code of the program  
        System.out.println("rest of the code...");  
    }  
}
```

# Examples

```
class JavaException {  
    public static void main(String args[]) {  
        int d = 0;  
        int n = 20;  
        try {  
            int fraction = n / d;  
            System.out.println("This line will not be Executed");  
        }  
        catch (ArithmaticException e) {  
            System.out.println("In the catch Block due to Exception = " + e);  
        }  
        System.out.println("End Of Main");  
    }  
}
```



**University Academy**

Teaching|Training|Informative

# **Web Technology**

## **(Java programming)**

**Unit-I**

### ***Java Interfaces***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Introduction**
- **Why interface**
- **Defining a Interfaces**
- **Example**

# Introduction

- An **interface in java** is a blueprint of a class. It has static constants and abstract methods.
- Interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body.
- It is used to achieve abstraction and multiple inheritance in Java
- Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body

# Why interface

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.
- This is the general form of an interface:

```
access interface name {  
    return-type method-name1(parameter-list);  
    return-type method-name2(parameter-list);  
    type final-varname1 = value;  
    type final-varname2 = value;  
    // ...  
    return-type method-nameN(parameter-list);  
    type final-varnameN = value;  
}
```

# Example

```
interface printable{  
    void print();  
}  
  
class A6 implements printable{  
    public void print(){System.out.println("Hello");}  
    public void show(){System.out.println("How r you");}  
    public static void main(String args[]){  
        printable obj=new A6();  
        A6 obj1 = new A6();  
        obj.print();  
        obj1.show();  
    }  
}
```

# Multiple inheritance in Java by interface

```
interface Printable{  
    void print();  
}  
interface Showable{  
    void show();  
}  
class A7 implements Printable,Showable{  
    public void print(){System.out.println("Hello");}  
    public void show(){System.out.println("Welcome");}  
    public static void main(String args[]){  
        A7 obj = new A7();  
        obj.print();  
        obj.show(); }  
}
```



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-I**

***Java I/O***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

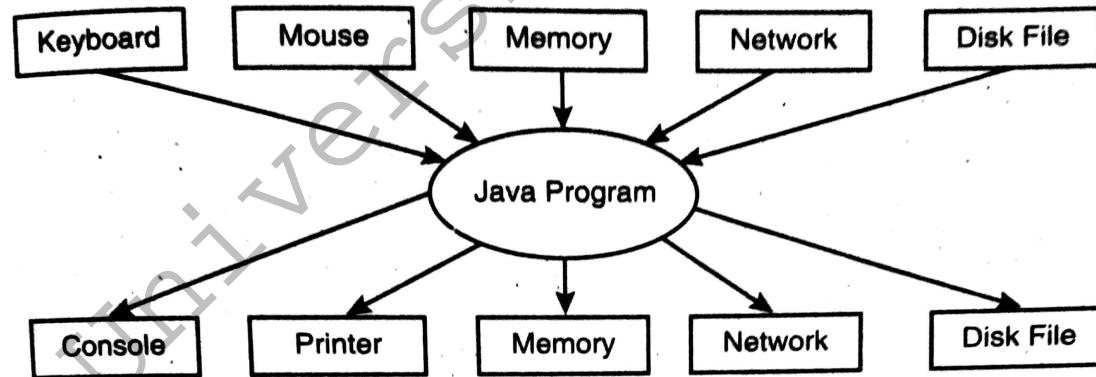
**Technical University,Lucknow**

# Summary

- **Introduction**
- **Stream**
- **Java Stream Classes**
- **Examples**

# Introduction

- Java I/O (Input and Output) is used to process the input and produce the output.
- Java uses the concept of a *stream* to make I/O operation fast.
- The `java.io` package contains all the classes required for input and output operations.
- We can perform **file handling in Java** by Java I/O API.

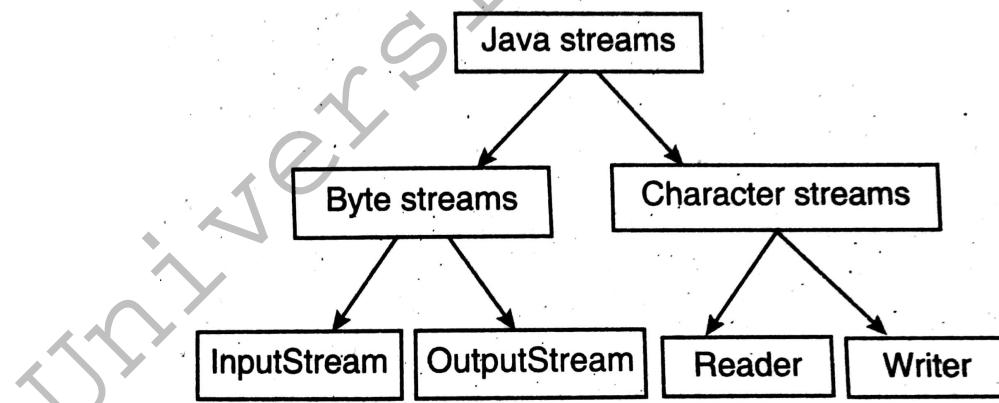


# Stream

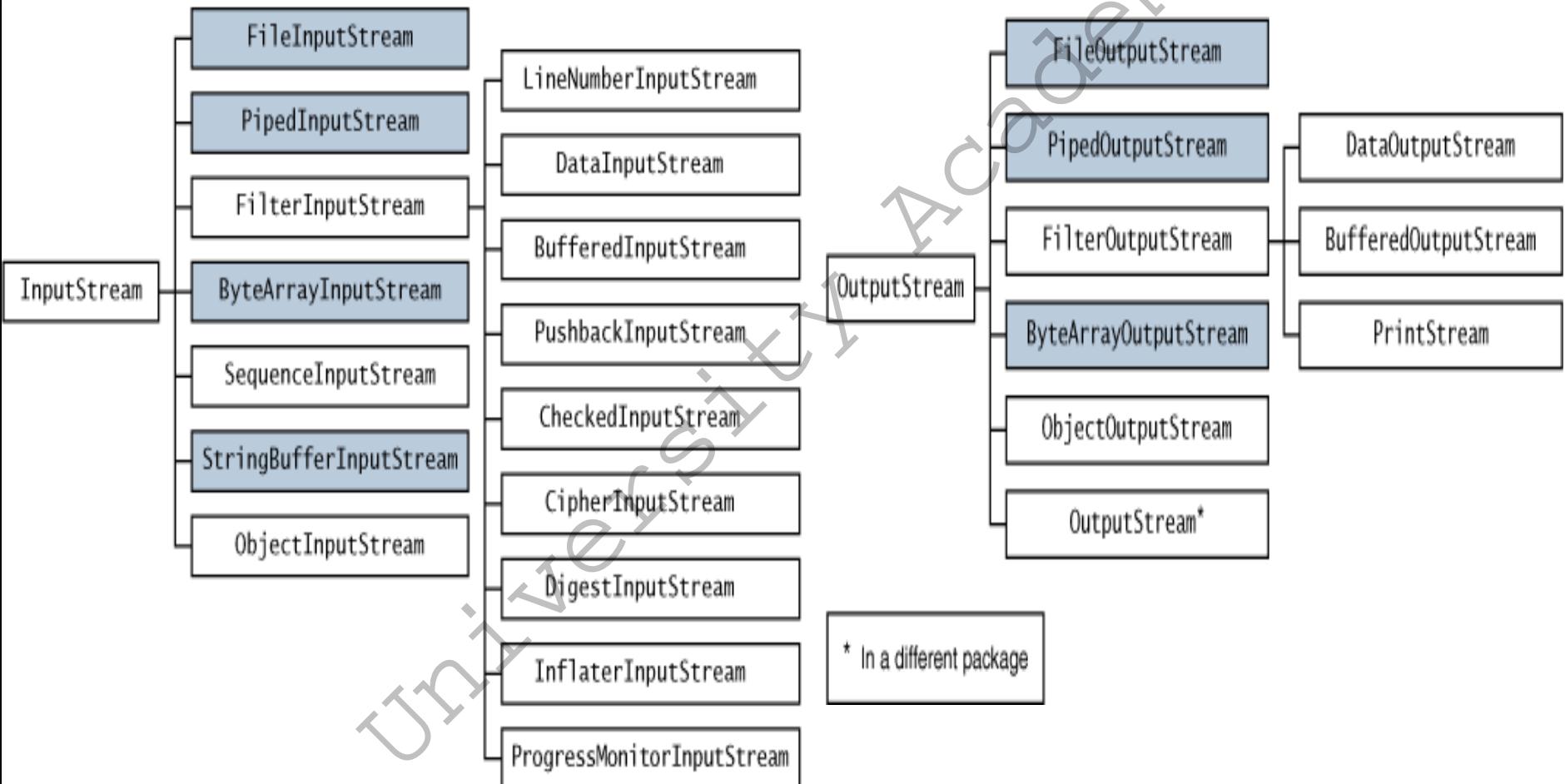
- A Stream is an ordered sequence of bytes that has an input stream and an output stream.
- In Java, 3 streams are created for us automatically. All these streams are attached with the console.
  - **System.out:** standard output stream
  - **System.in:** standard input stream
  - **System.err:** standard error stream
- Java implements *streams* within class hierarchies defined in the **java.io package**.

# Java Stream Classes

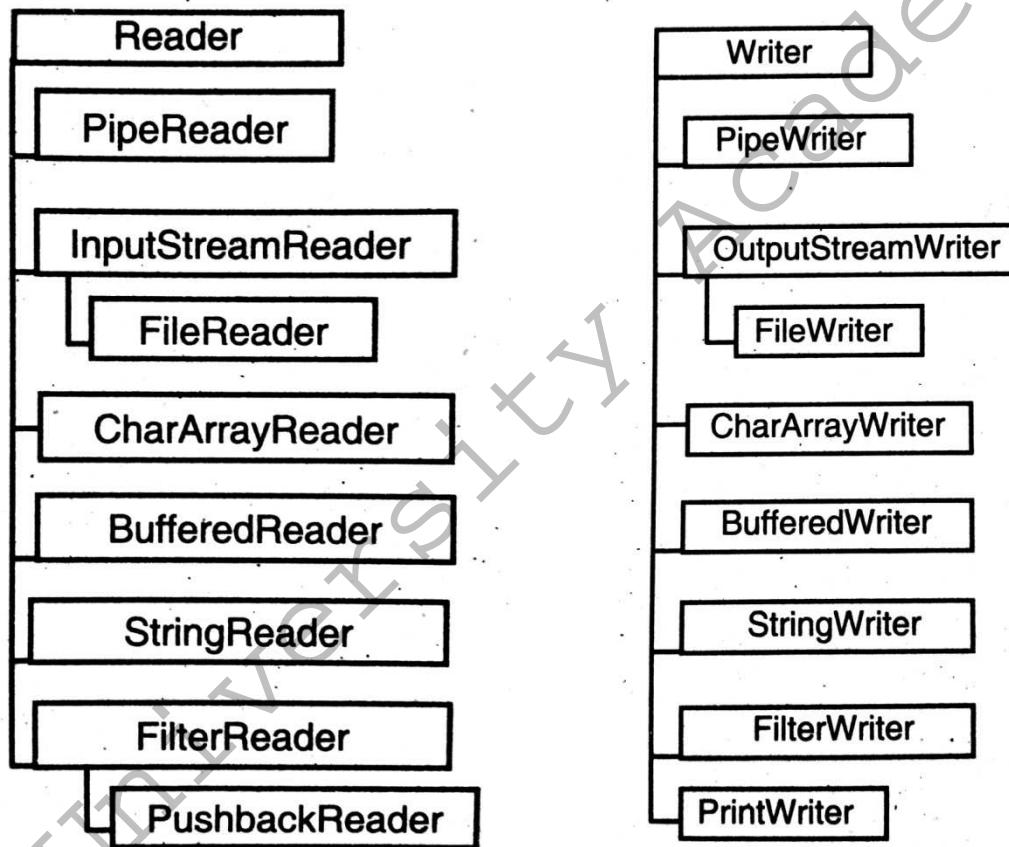
- Java defines two types of streams
  - **Byte Streams** : provide a convenient means for handling input and output of bytes. Byte streams are used, for example, when reading or writing binary data.
  - **Character Streams** : provide a convenient means for handling input and output of characters. They use Unicode.



# Byte Stream Classes



# Character Streams Classes



# Byte Stream Example

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
        try{
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
            fout.write(65);
            fout.close();
            System.out.println("success...");
        }catch(Exception e){System.out.println(e);}
    }
}
```

# Byte Stream Example

```
import java.io.FileOutputStream;
public class FileOutputStreamExample1 {
    public static void main(String args[]){
        try{
            FileOutputStream fout=new FileOutputStream("D:\\testout1.txt");
            String s="Welcome to java.";
            byte b[]=s.getBytes();//converting string into byte array
            fout.write(b);
            fout.close();
            System.out.println("success...");
        }catch(Exception e){System.out.println(e);}
    }
}
```

# Byte Stream Example

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
            FileInputStream fin=new FileInputStream("D:\\testout.txt");
            int i=fin.read();
            System.out.print((char)i);

            fin.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

# Character Streams Examples

```
import java.io.*;
public class WriterExample {
    public static void main(String[] args) {
        try {
            Writer w = new FileWriter("D:\\testout2.txt");
            String content = "I love my country";
            w.write(content);
            w.close();
            System.out.println("Done");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# Character Streams Examples

```
import java.io.*;
public class ReaderExample {
    public static void main(String[] args) {
        try {
            Reader reader = new FileReader("D:\\testout2.txt");
            int data = reader.read();
            while (data != -1) {
                System.out.print((char) data);
                data = reader.read();
            }
            reader.close();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```



**University Academy**

Teaching|Training|Informative

# **Web Technology**

## **(Java programming)**

**Unit-I**

### ***Multithreading in Java***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Introduction**
- **Advantage**
- **Life cycle of a Thread**
- **Creating a Thread**
- **Examples**

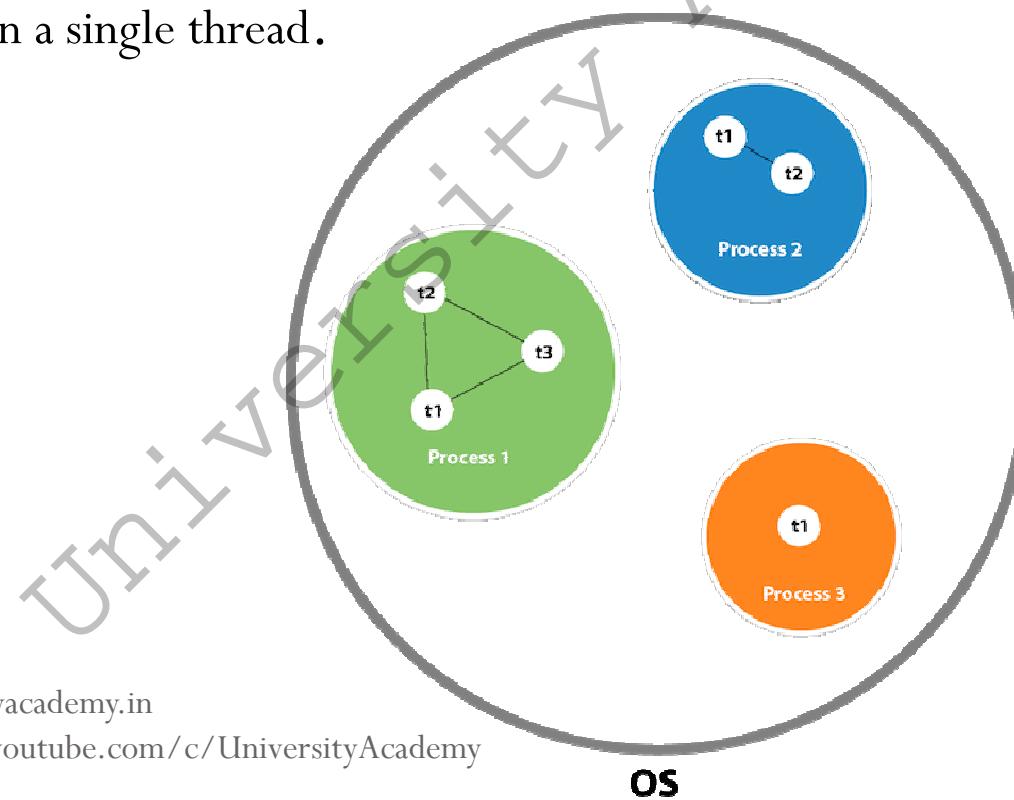
# Introduction

- **Multithreading in java** is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing.
- Multiprocessing and multithreading, both are used to achieve multitasking.

Multitasking	Multithreading	Multiprocessing
<ul style="list-style-type: none"><li>• Capacity to run more than one process concurrently in a single CPU environment.</li><li>• Multitasking may be Process-based or Thread-based</li></ul>	<ul style="list-style-type: none"><li>• A single process that contains more than one thread (sub-processes) running simultaneously is called as a Multithread.</li><li>• It is a thread-based multitasking</li><li>• use a shared memory area</li></ul>	<ul style="list-style-type: none"><li>• It is much like multitasking doing with more than one CPU.</li><li>• It is a process-based multitasking</li><li>• separate memory area</li></ul>

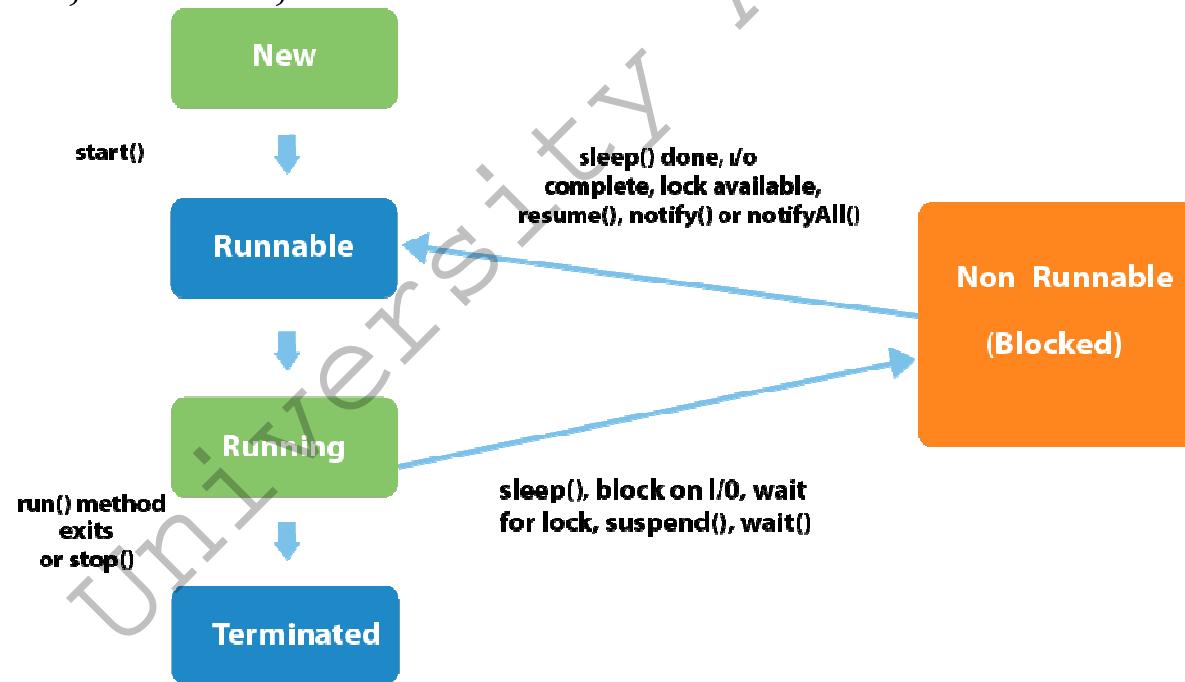
# Advantage

- **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
- You **can perform many operations together, so it saves time.**
- Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.



# Life cycle of a Thread

- A thread can be in one of the five states: *New, Runnable, Running, Non-Runnable (Blocked), Terminated.*
- According to sun, there is only 4 states in thread life cycle in java *new, runnable, non-runnable and terminated.*



# Creating Thread

- Threads can be created by using two mechanisms :
  - **Extending the Thread class**
  - **Implementing the Runnable Interface**
- Commonly used Constructors of Thread class:
  - Thread()
  - Thread(String name)
  - Thread(Runnable r)
  - Thread(Runnable r, String name)
- Commonly used methods of Thread class:
  - public void run()
  - public void start()
  - public void sleep(long miliseconds)
  - public int getId()
  - public void resume()
  - public void stop()
  - public String getName()

# Extending the Thread class

- The Class which extends the *thread* class will become a sub class.
- The thread class is contained in the package *java.lang.Thread*
  - STEP 1: Create a class by extending a Thread class
  - STEP 2: Override the run() method of a Thread class and place it inside the class which extends the Thread class.
  - STEP 3: Create an object for the thread inside the main() method.
  - STEP 4: Invoke the start() method through the object of the Thread class to execute the run() method of a thread.

# Extending the Thread class

```
class Demo extends Thread  
{  
    public void run()  
    {  
        System.out.println("Thread started execution");  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println("Multithreading");  
        }  
        System.out.println("Thread is completed");  
    }  
}
```

```
public class Firstthread  
{  
    public static void main(String args[])  
    {  
        Demo t1=new Demo();  
        Demo t2=new Demo();  
        t1.start();  
        t2.start();  
    }  
}
```

# Implementing the Runnable Interface

- The simplest way of creating a thread is by implementing a *Runnable* interface.
- At a time any number of threads can be created in a program using *Runnable* interface.
- We create a new class which implements *java.lang.Runnable*
  - STEP 1: As a first step, the thread should be created through implementing a *Runnable* interface.
  - STEP 2: Define a *run()* method inside the thread. The *run()* method can be used only after implementing the *Runnable* interface.
  - STEP 3: Place the *run()* method inside the class which implements a *Runnable* interface.
  - STEP 4: Next is to instantiate a *Thread* class inside the *main()* method and use that object to execute a thread.

# Implementing the Runnable Interface

## Program 10.2

```
class RT implements Runnable
{
    public void run()
    {
        System.out.println("Thread started its
execution");
        for(int i=1;i<=10;i++)
        {
            System.out.println(i+" * 6 = "+i*6);
        }
        System.out.println("Thread completed its
execution");
    }
    public class Firstthread
    {
        public static void main(String[] args)
        {
            RT r=new RT();
            Thread t=new Thread(r);
            t.start();
        }
    }
}
```

## Output 10.2

```
Thread started its
execution
1 * 6 = 6
2 * 6 = 12
3 * 6 = 18
4 * 6 = 24
5 * 6 = 30
6 * 6 = 36
7 * 6 = 42
8 * 6 = 48
9 * 6 = 54
10 * 6 = 60
Thread completed its
execution
```



**University Academy**  
Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-I**

## ***Java Package***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

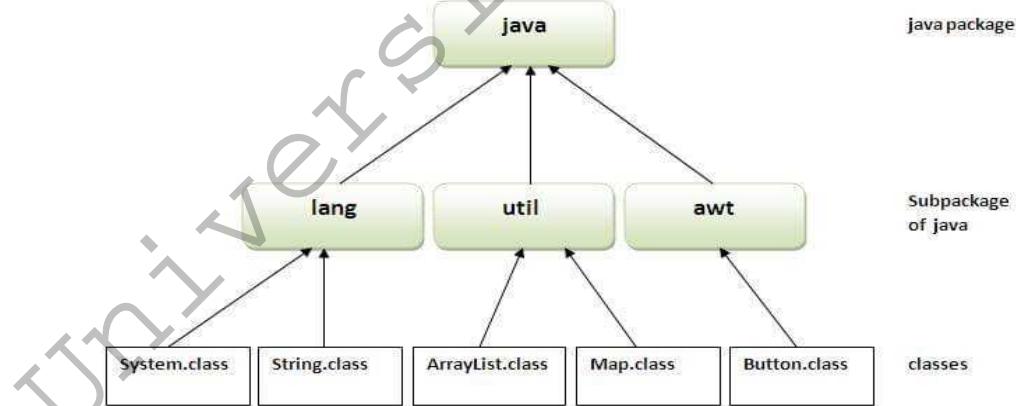
- **Packages Introduction**
- **Advantage**
- **Defining a Package**
- **Package Example**
- **Importing Packages**
- **Access Specifiers**

# Packages Introduction

- A java package is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form,
  - built-in package and
  - user-defined package
- built-in packages such as java,lang, awt, javax, swing, net, io, util, sql etc.

# Advantage

- Advantage of Java Package
  - Java package is used to categorize the classes and interfaces so that they can be easily maintained.
  - Java package provides access protection.
  - Java package removes naming collision.



# Defining a Package

- Simply include a package command as the first statement in a Java source file. This is the general form of the package statement:

*package pkg;*

- Eg: *package MyPack;*
- You can create a hierarchy of packages For example, a package declared as

*package java.awt.image;*

- In order for a program to find MyPack, one of three things must be true.
  - Either the program can be executed from a directory immediately above MyPack,
  - or the CLASSPATH must be set to include the path to MyPack,
  - or the -classpath option must specify the path to MyPack when the program is run via java.

# Package Example

```
// A simple package  
package MyPack;  
class Balance {  
    String name;  
    double bal;  
    Balance(String n, double b) {  
        name = n;  
        bal = b;  
    }  
    void show() {  
        if(bal<0)  
            System.out.print("--> ");  
        System.out.println(name + ": $" + bal);  
    }  
}
```

```
class AccountBalance {  
    public static void main(String args[]) {  
        Balance current[] = new Balance[3];  
        current[0] = new Balance("K. J. Fielding",  
                               123.23);  
        current[1] = new Balance("Will Tell", 157.02);  
        current[2] = new Balance("Tom Jackson", -  
                               12.33);  
        for(int i=0; i<3; i++) current[i].show();  
    }  
}
```

# Importing Packages

- There are three ways to Import package.
  - import package.\*;
  - import package.classname;
  - fully qualified name.

```
// save by A.java
package Pack;
public class A{
    public void
msg(){System.out.println("Hello");
}
}
```

```
// save by B.java
import Pack.*;
class B{
    public static void main(String args[]){
        A obj = new A();
        obj.msg();
    }
}
```

# Access Specifiers

TABLE 9-1  
Class Member  
Access

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes



**University Academy**  
Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-I**

## ***String Handling***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Introduction**
- **String Methods**
- **Examples**

# Introduction

- String is a collection characters.
- String is basically an object that represents sequence of char values.
- An array of characters works same as Java string
- Example

```
char[] ch={'U','n','i','v','e','r','s','i','t','y'};  
String s=new String(ch);  
OR  
String s="University";
```

# String Methods

Method	Description
s1.charAt(position)	Returns the character present at the index <i>position</i> .
s1.compareTo(s2)	If $s1 < s2$ then it returns positive, if $s1 > s2$ then it returns negative and if $s1 = s2$ then it returns zero.
s1.concat(s2)	It returns the concatenated string of s1 and s2.
s1.equals(s2)	If s1 and s2 are both equal then it returns true.
s1.equalsIgnoreCase(s2)	By ignoring case, if s1 and s2 are equal then it returns true.
s1.indexOf('c')	It returns the first occurrence of character 'c' in the string s1.
s1.indexOf('c',n)	It returns the position of 'c' that occur at after nth position in string s1.
s1.length()	It gives the length of string s1.
String.valueOf(var)	Converts the value of the variable passed to it into String type.

# Example

```
public class StringExample{  
    public static void main(String args[]){  
        String s1="java";//creating string by java string literal  
        char ch[]={'s','t','r','i','n','g','s'};  
        String s2=new String(ch);//converting char array to string  
        String s3=new String("example");//creating java string by new keyword  
        System.out.println(s1);  
        System.out.println(s2);  
        System.out.println(s3);  
        int x=s2.length();  
        System.out.println(x);  
        boolean b=s1.equals(s2);  
        System.out.println(b);  
        s2=s2.concat(s3);  
        System.out.println(s2);  
    }  
}
```

Lecture Notes  
On  
**Web Technology**  
  
*By*  
**Mr. Sandeep Vishwakarma**  
*Assistant Professor*  
**Dr. A.P.J. Abdul Kalam**  
*Technical University,Lucknow*

---

**University Academy**  
**Teaching | Training | Informative**

Email: universityacademy.in@gmail.com, Website: www.universityacademy.in,  
YouTube: www.youtube.com/c/UniversityAcademy, Contact: +91-9411988382, +91-989

## **Web Technology**

### **Unit-II**

#### **HTML:**

HTML stands for Hyper Text Markup Language .HTML is a technique to design web pages. HTML is not a programming languages, it is markup languages. A markup language is set of markup tag and used these tag to describe web pages.

#### **Versions of HTML**

HTML : In 1991 by Berners-Lee

HTML 2.0 : in 1995 by T. Berners-Lee and D. Connolly

HTML 3.2: in 1997 together with vendors including IBM, Microsoft, Netscape Communications Corporation, Novell, SoftQuad, Spyglass, and Sun Microsystems.

HTML 4.01: in 1999 by recommended by the W3C in December

HTML5 : In 2014 has been developed by two separate groups: the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG)

#### **Examples:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>University Academy</title>
  </head>
  <body>
    <h1>Welcome to My Site </h1>
    <p>About WebSite ....</p>
  </body>
</html>
```

Write code in notepad and save with .html or .htm extension

Output:



## Welcome to My Site

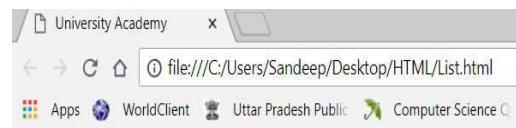
About WebSite ....

<!DOCTYPE...> This tag defines the document type and HTML version  
<html>: Describe the web page and mainly comprises of document header which is represented by <head>...</head> & document body which is represented by <body>...</body> tags.  
<head>: This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.  
<title> The <title> tag is used inside the <head> tag to mention the document title.  
<body> This tag represents the document's body which keeps other HTML tags like <h1>, <div>, <p> etc.  
<h1> this tag represents the heading.  
<p> this tag represents a paragraph.

## HTML list

HTML lists are defined with the `<ul>` (unordered/bullet list) or the `<ol>` (ordered/numbered list) tag, followed by `<li>` tags (list items)

```
<!DOCTYPE html>
<html>
<head>
<title>University Academy</title>
</head>
<body>
<h2>An unordered HTML list</h2>
<ul>
    <li>Coffee</li>
    <li>Tea</li>
    <li>Milk</li>
</ul>
<h2> An Ordered HTML list </h2>
<ol>
    <li>Coffee</li>
    <li>Tea</li>
    <li>Milk</li>
</ol>
</body>
</html>
```



An unordered HTML list

- Coffee
- Tea
- Milk

An Ordered HTML list

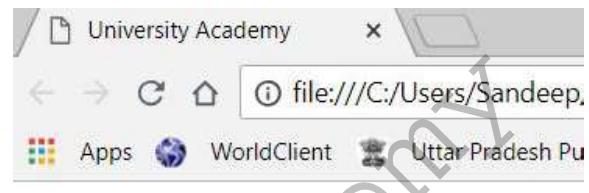
1. Coffee
2. Tea
3. Milk

## HTML table

Tables are the collection of row and columns. Tables are defined within the tag <table>. Table is divided into row <tr> and each row divided into column <td> . we can enter the data in to cell using <td>.the data can text, image, list form, table etc.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>University Academy</title>
</head>
<body>
    <table>
        <tr>
            <th>Firstname</th>
            <th>Lastname</th>
            <th>Age</th>
        </tr>
        <tr>
            <td>Sandeep</td>
            <td>Vishwakarma</td>
            <td>31</td>
        </tr>
        <tr>
            <td>Ravi</td>
            <td>Maurya</td>
            <td>32</td>
        </tr>
    </table>
</body>
</html>
```



Firstname	Lastname	Age
Sandeep	Vishwakarma	31
Ravi	Maurya	32

## HTML images

Images can improve the design and the appearance of a web page. Images are defined with the tag <img> that is empty means it contain attribute only and it has no closing tag.

To display the image on the pages we uses src attribute.

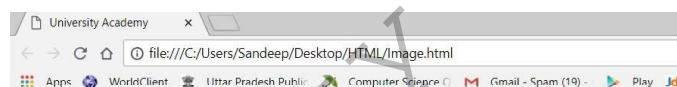
Syntax: 

Example

```
<!DOCTYPE html>
<html>
<head>
<title>University Academy</title>
</head>

<body>

</body>
</html>
```



### Components of Image:

- Width
- Height
- Hspace
- Vspace
- Align
- Alt

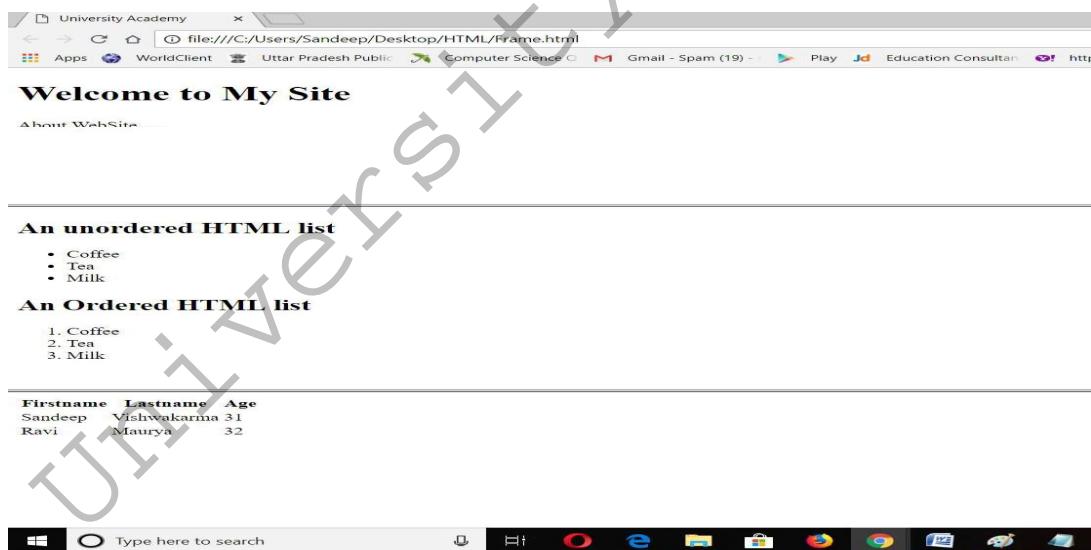
## HTML frames

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>University Academy</title>
  </head>
  <frameset rows = "30%,40%,30%">
    <frame name ="top" src = "C:\Users\Sandeep\Desktop\HTML\Home.html" />
    <frame name ="main" src ="C:\Users\Sandeep\Desktop\HTML\List.html">
    <frame name ="bottom" src = "C:\Users\Sandeep\Desktop\HTML\Table.html" />
  </frameset>

  <body>
  </body>
</html>
```



## HTML forms

HTML Forms are required, when you want to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML **<form>** tag is used to create an HTML form

Syntax

```
<form>
  .
  .
</form>
```

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>University Academy</title>
</head>
<body>
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
<form>
  <input type="radio" name="gender" value="male" checked>
  Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other
</form>
<form name="input"
action="C:\Users\Sandeep\Desktop\HTML\Home.html",
method="get">
  <input type="submit",value="submit",>
```

```
</form>
</body>
</html>
```

The screenshot shows a Microsoft Edge browser window titled "University Academy". The address bar displays "file:///C:/Users/Sandeep/Desktop/HTML/Form.html?". The page content contains a form with fields for "First name:" and "Last name:", each with an associated text input box. Below these fields are three radio buttons labeled "Male", "Female", and "Other". A "Submit" button is located at the bottom right of the form area.



About WebSite ....

## CSS

CSS stands for Cascading Style Sheets. CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**. W3C has actively promoted the use of style sheets on the Web since the consortium was founded in 1994. Cascading Style Sheets (CSS) provide easy and effective alternatives to specify various attributes for the HTML tags.

SS can be added to HTML elements in 3 ways:

- **Inline** - by using the style attribute in HTML elements
- **Internal** - by using a `<style>` element in the `<head>` section
- **External** - by using an external CSS file

*Inline:*

```
<!DOCTYPE html>
<html>
<head>
<title>University Academy</title>
</head>
<body>
<h1 style="color:blue;">Welcome to My Site </h1>
<p style="color:red;">About WebSite ....</p>
</body>
</html>
```



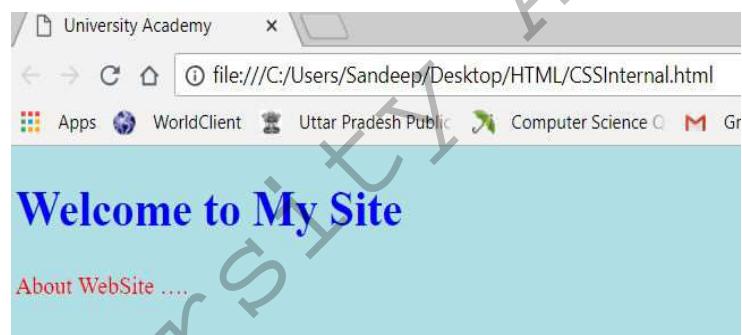
## Welcome to My Site

About WebSite ....

*Internal:* Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>University Academy</title>
    <style>
      body {background-color: powderblue;}
      h1 {color: blue;}
      p {color: red;}
    </style>
  </head>
  <body>
    <h1>Welcome to My Site </h1>
    <p>About WebSite ....</p>
  </body>
</html>
```

Output:



**External:** Example save this file in .html extension

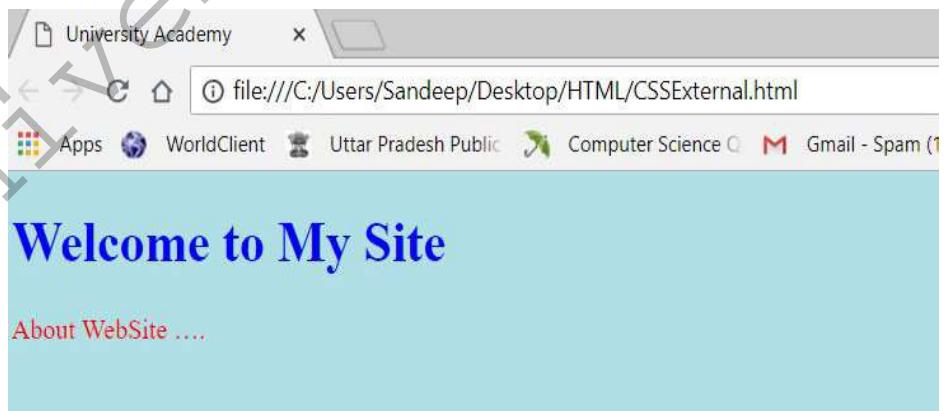
```
<!DOCTYPE
html>

<html>
<head>
<title>University Academy</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h1>Welcome to My Site </h1>
<p>About WebSite ....</p>
</body>
</html>
```

**CSS file:** save this file in .css extension

```
body {
    background-color: powderblue;
}
h1 {
    color: blue;
}
p {
    color: red;
}
```

**Output:**



## XML:

DTD, XML schemes, Object Models, presenting and using XML, Using XML Processors: DOM and SAX, Dynamic HTML

XML stands for **Extensible Markup Language** and is a text-based markup language derived from Standard Generalized Markup Language (SGML). XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

### **HTML vs XML**

There are many differences between HTML (Hyper Text Markup Language) and XML (eXtensible Markup Language). The important differences are given below:

No.	HTML	XML
1)	HTML is used <b>to display data</b> and focuses on how data looks.	XML is a software and hardware independent tool used <b>to transport and store data</b> . It focuses on what data is.
2)	HTML is a <b>markup language</b> itself.	XML provides a <b>framework to define markup languages</b> .
3)	HTML is <b>not case sensitive</b> .	XML is <b>case sensitive</b> .
4)	HTML is a presentation language.	XML is neither a presentation language nor a programming language.
5)	HTML <b>has its own predefined tags</b> .	You <b>can define tags according to your need</b> .
6)	In HTML, it is <b>not necessary to use a closing tag</b> .	XML <b>makes it mandatory to use a closing tag</b> .
7)	HTML is <b>static</b> because it is used to display data.	XML is <b>dynamic</b> because it is used to transport data.
8)	HTML <b>does not preserve whitespaces</b> .	XML <b>preserve whitespaces</b> .

## XML Tools

**XML – Viewers:** An XML document can be viewed using a simple text editor or any browser. XML files are saved with a ".xml" extension

**XML – Editors:** XML Editor is a markup language editor. The XML documents can be edited or created using existing editors such as Notepad, WordPad, or any similar text editor. You can also find a professional XML editor Eg. Xerlin, CAM - Content Assembly Mechanism.

**XML – Parsers:** **XML parser** is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers.

**XML – Processors:** When a software program reads an XML document and takes actions accordingly, this is called processing the XML. Any program that can read and process XML documents is known as an XML processor. An XML processor reads the XML file and turns it into in-memory structures that the rest of the program can access

### XML Example: contact.xml

```
<?xml version = "1.0"?>
```

```
<contact-info>
```

```
    <name>Tanmay Patil</name>
```

```
    <company>TutorialsPoint</company>
```

```
    <phone>(011) 123-4567 </phone>
```

```
</contact-info>
```



## XML Validation

XML file can be validated by 2 ways define XML structure.

1. against DTD (Document Type Definition)
2. against XSD XML Schema Definition

**The XML Document Type Declaration**, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

## Syntax

Basic syntax of a DTD is as follows –

```
<!DOCTYPE element DTD identifier  
[  
    declaration1  
    declaration2  
    .....  
]>
```

- The **DTD** starts with `<!DOCTYPE` delimiter.
- An **element** tells the parser to parse the document from the specified root element
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. it is called *External Subset*.
- **The square brackets [ ]** enclose an optional list of entity declarations called *Internal Subset*.

Following is a simple example of internal DTD –

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>  
<!DOCTYPE address [  
    <!ELEMENT address (name,company,phone)>  
    <!ELEMENT name (#PCDATA)>  
    <!ELEMENT company (#PCDATA)>  
    <!ELEMENT phone (#PCDATA)>  
>  
<address>  
    <name>Tanmay Patil</name>  
    <company>TutorialsPoint</company>  
    <phone>(011) 123-4567</phone>  
</address>
```

The following example shows external DTD usage –

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>  
<!DOCTYPE address SYSTEM "address.dtd">
```

```
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** is as shown –

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-I**

**XML**

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Introduction XML**
- **Characteristics of XML**
- **XML - Syntax**
- **HTML vs XML**
- **XML Validation**

# Introduction

- XML stands for Extensible Markup Language. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).
- SGML is a meta-markup language.
- A Meta-Markup Language is a language that means it allows us to create our own tags.
- XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data

# Characteristics of XML

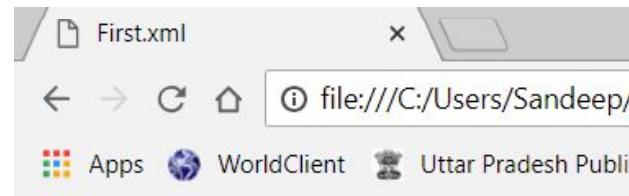
- **XML is extensible** – XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it** – XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard** – XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

# XML - Syntax

```
<?xml version = "1.0"?>  
<contact-info>  
    <name>Sandeep Vishwakarma</name>  
    <company>RKGIT</company>  
    <phone>9411988382</phone>  
</contact-info>
```

Save XML document as first.xml  
and open it in browsers

- ✓ XML Declaration
- ✓ Tags and Elements
- ✓ Element Syntax
- ✓ Nesting of Elements
- ✓ Root Element
- ✓ Case Sensitivity



This XML file does not appear to have any st

```
▼<contact-info>  
    <name>Sandeep Vishwakarma</name>  
    <company>RKGIT</company>  
    <phone>9411988382</phone>  
</contact-info>
```

# HTML vs XML

No.	HTML	XML
1)	HTML is used <b>to display data</b> and focuses on how data looks.	XML is a software and hardware independent tool used <b>to transport and store data</b> . It focuses on what data is.
2)	HTML is a <b>markup language</b> itself.	XML provides a <b>framework to define markup languages</b> .
3)	HTML is <b>not case sensitive</b> .	XML is <b>case sensitive</b> .
4)	HTML is a presentation language.	XML is neither a presentation language nor a programming language.
5)	HTML <b>has its own predefined tags</b> .	You <b>can define tags according to your need</b> .
6)	In HTML, it is <b>not necessary to use a closing tag</b> .	XML <b>makes it mandatory to use a closing tag</b> .
7)	HTML is <b>static</b> because it is used to display data.	XML is <b>dynamic</b> because it is used to transport data.

# XML Validation

- XML file can be validated by 2 ways define XML structure.
  - Against DTD (Document Type Definition)
  - Against XSD (XML Schema Definition)
- DTD (Document Type Definition):
  - DTD is technique is used to define the structure of XML Document
  - DTD is text based document with .dtd extension
  - DTD contain *element declaration*, *Attribute Declaration*, *Entity references declaration*
    - *Element declaration*: <!ELEMENT element-name (content-model)>
    - *Attribute Declaration*: <!ATTLIST element-name attribute-name attribute-type attribute-value>
    - *Entity references declaration*: <!ENTITY entity\_name "entity\_value">
  - Eg <!ELEMENT address (name,company,phone)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT company (#PCDATA)> //parse able character data  
<!ATTLIST name id CDATA #REQUIRED>  
<!ENTITY phone "(011) 123-4567">

# Syntax

```
<!DOCTYPE element  
[  
Declaration1  
declaration2 .....  
]>
```

Example:

```
<!DOCTYPE address  
[  
<!ELEMENT address (name,company,phone)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT company (#PCDATA)>  
<!ELEMENT phone (#PCDATA)>  
]>
```

# Types of DTD

- ***Internal DTD:*** A DTD is referred to as an internal DTD if elements are declared within the XML files.

```
<?xml version = "1.0">
<!DOCTYPE address
[
    <!ELEMENT address (name,company,phone)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT company (#PCDATA)>
    <!ELEMENT phone (#PCDATA)> ]>
<address>
    <name>Sandeep</name>
    <company>RKGIT</company>
    <phone>9411988382</phone>
</address>
```

# External DTD

- In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL.

```
<!ELEMENT address(name,company,phone)>
<!ELEMENT name(#PCDATA)>
<!ELEMENT company(#PCDATA)>
<!ELEMENT phone(#PCDATA)>
```

// Save as address.dtd

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
    <name>Sandeep</name>
    <company>RKGIT</company>
    <phone>9411988382</phone>
</address>
```

// Save this file in .xml extention

# XSD (XML Schema Definition)

- XML Schema is commonly known as **XML Schema Definition (XSD)**.
- It is used to describe and validate the structure and the content of XML data.
- XML schema defines the elements, attributes and data types. Schema element supports Namespaces.
- It is similar to a database schema that describes the data in a database.

# Definition Types

- **Simple Type:** Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date.
- For example <xs:element name = "phone\_number" type = "xs:int" />
- **Complex Type:** A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents

```
<xs:element name = "Address">
    <xs:complexType>
        <xs:sequence>
            <xs:element name = "name" type = "xs:string" />
            <xs:element name = "company" type = "xs:string" />
            <xs:element name = "phone" type = "xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

# Example

## XML Document with DTD

**student.dtd**

```
<!ELEMENT student (name,address,std,marks)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT std (#PCDATA)>
<!ELEMENT marks (#PCDATA)>
```

**DTDDemo.xml**

```
<?xml version="1.0"?>
<!DOCTYPE student SYSTEM "student.dtd">
<student>
    <name>Anand</name>
    <address>Pune</address>
    <std>Second</std>
    <marks>70 percent</marks>
</student>
```

# Example

## XML Schema

### **StudentSchema.xsd**

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Student">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="address" type="xs:string"/>
                <xs:element name="std" type="xs:string"/>
                <xs:element name="marks" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

### **MySchema.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<Student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="StudentSchema.xsd">
    <name>Anand</name>
    <address>Pune</address>
    <std>Second</std>
    <marks>70 percent</marks>
</Student>
```



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-III**

## ***Java Script***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Introduction**
- **Java Vs JavaScript**
- **JavaScript - Syntax**
- **JavaScript – Placement**
- **Examples**
- **Important Questions**

# Introduction

- Javascript is a dynamic computer programming language.
- It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.
- It is an interpreted programming language with object-oriented capabilities.
- Java Script was developed by Netscape in 1995.
- JavaScript was first known as **LiveScript**, but Netscape changed its name to JavaScript.

# Introduction

- Java Script has three part.
  - **Core:** it include Operators, Expressions, Statements, and Subprogram.
  - **Client Side:** It is a collection of object using which one can control over the browser.
  - **Server Side :** It is a collection of object using which one can access the database on the server.

# Java Vs JavaScript

## JAVA

Java is strongly typed language and variable must be declare first to use in program. In Java the type of a variable is checked at compile-time.

Java is an object oriented programming language.

Java applications can run in any virtual machine(JVM) or browser.

Objects of Java are class based even we can't make any program in java without creating a class.

## JAVASCRIPT

JavaScript is weakly typed language and have more relaxed syntax and rules.

JavaScript is an object based scripting language.

JavaScript code run on browser only as JavaScript is developed for browser only.

JavaScript Objects are prototype based.

# Java Vs JavaScript

## JAVA

Java program has file extension “.Java” and translates source code into bytecodes which is executed by JVM(Java Virtual Machine).

Java is a Standalone language.

Java program uses more memory.

Java has a thread based approach to concurrency.

## JAVASCRIPT

JavaScript file has file extension “.js” and it is interpreted but not compiled, every browser has the Javascript interpreter to execute JS code.

contained within a web page and integrates with its HTML content.

JavaScript requires less memory therefore it is used in web pages.

Javascript has event based approach to concurrency.

# JavaScript - Syntax

- JavaScript can be implemented using JavaScript statements that are placed within the `<script>...</script>`.
- You can place the `<script>` tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the `<head>` tags.

```
<script ...>  
    JavaScript code  
</script>
```

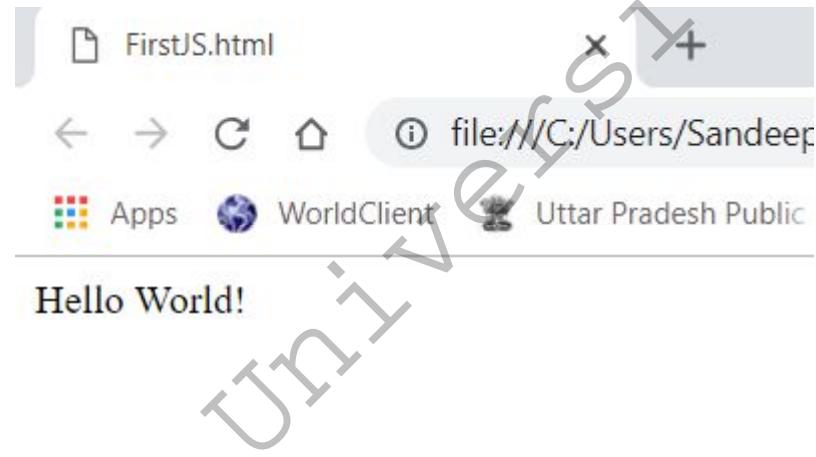
```
<script language="javascript" type="text/javascript">  
    JavaScript code  
</script>
```

```
<script language="javascript" type="text/javascript" src=MyPage.js>  
    JavaScript code  
</script>
```

- **Language** – This attribute specifies what scripting language you are using. Typically, its value will be javascript.
- **Type** – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

# Example

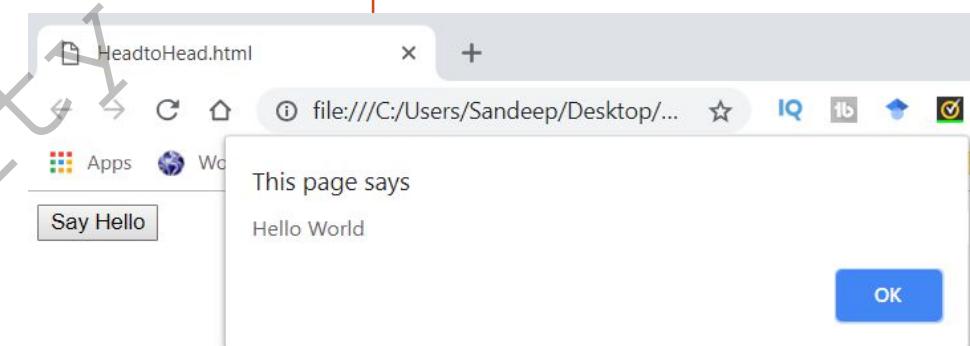
```
<html>
  <body>
    <script language="javascript" type="text/javascript">
      document.write("Hello World!")
    </script>
  </body>
</html>
```



# JavaScript - Placement

- Most preferred ways to include JavaScript in an HTML file are as follows –
  - *Script in <head>...</head> section.*
  - Script in <body>...</body> section.
  - Script in <body>...</body> and <head>...</head> sections.
  - Script in an external file and then include in <head>...</head> section.

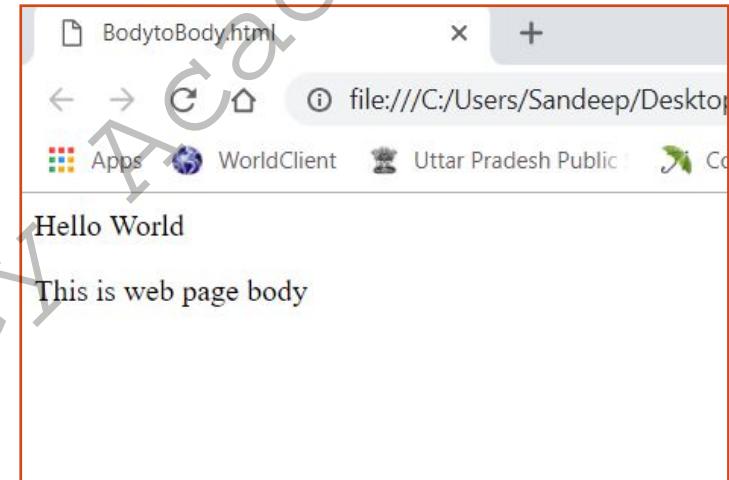
```
<html>
<head>
<script type="text/javascript">
function sayHello() {
    alert("Hello World") }
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```



# JavaScript - Placement

- *JavaScript in <body>...</body> section*

```
<html>
<head>
</head>
<body>
    <script type="text/javascript">
        document.write("Hello World")
    </script>
    <p>This is web page body </p>
</body>
</html>
```

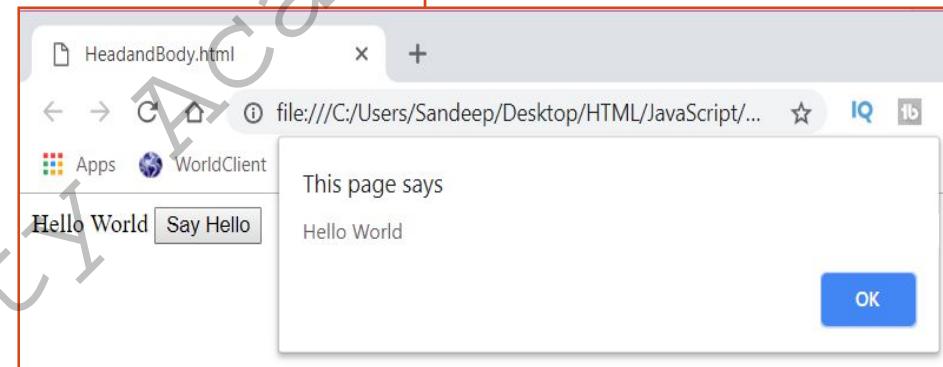


# JavaScript - Placement

- *JavaScript in <body> and <head> Sections*

```
<html>
<head>
    <script type="text/javascript">
        function sayHello() {
            alert("Hello World")
        }
    </script>
</head>
<body>
    <script type="text/javascript">
        document.write("Hello World")
    </script>

    <input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```



# JavaScript - Placement

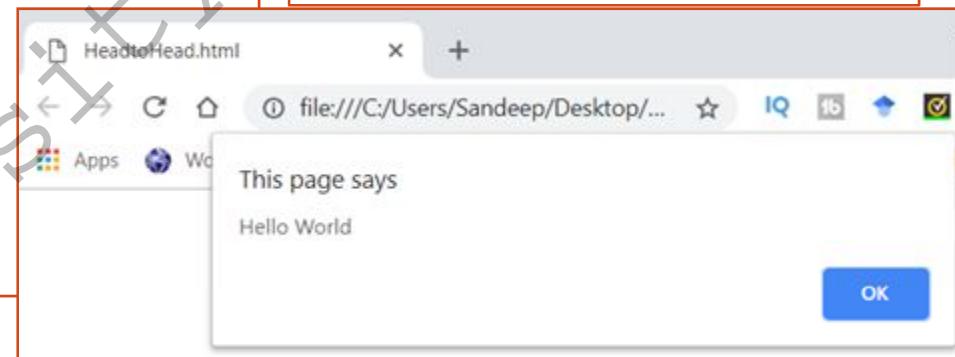
- *JavaScript in External File*

```
<html>
<head>
<script type="text/javascript" src="filename.js" >
</script>
</head>
<body> .....
</body>
</html>

// Save this file in .html Extension
```

```
function sayHello() {
    alert("Hello World")
}

// Save this file as filename.js
```



# *Important Questions ?*

1. What are difference between java and Java Script ?  
Describe the strength and weakness of java script.
2. What are the advantage of indirectly embedding the java script.
3. What is Java Script? Give an Example.



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-III**

## ***Java Script: Variables & Data Type***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Java Script Variables**
- **Java Script Data Types**
- **Java Script Reserved Words**
- **Examples**
- **Important Questions?**

# Variable Names

- They must begin with a letter or an underscore character or dollar sign.
- JavaScript variable names are case-sensitive.
- You should not use any of the JavaScript reserved keywords
- No Limit on length of identifiers
- Eg: **\_123test, myVar, name, money etc**

# Data Type

- JavaScript allows you to work with three primitive data types
  - **Numbers**, eg. 123, 120.50 etc.
  - **Strings** of text e.g. "This text string" etc.
  - **Boolean** e.g. true or false.
- JavaScript also defines two trivial data type each of which defines only a single value.
  - **null**
  - **Undefined**.
- In java script we can declare the variable using reserved word **var**. The value of this variable can be anything. Eg **Number, String , Boolean**.

# Variable Scope

- JavaScript Variable Scope
  - Global
  - Local

```
<html>
<body onload = checkscope();>
<script type = "text/javascript">
    <!-- var myVar = "global"; // Declare a global variable
        function checkscope( ) {
            var myVar = "local"; // Declare a local variable
            document.write(myVar); } //-->
</script> </body>
```

# JavaScript Reserved Words

- JavaScript Reserved Words: They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

# Example

```
<!DOCTYPE html>
<html>
    <body>
        <h2>JavaScript Variables</h2>
        <p>In this example, x, y, and z are variables.</p>
        <p id="demo"></p>
        <script>
            var x = 5;
            var y = 6;
            var z = x + y;
            document.getElementById("demo").innerHTML = "The value of z is: " + z;
        </script>
    </body>
</html>
```



# Example

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Variables</h2>
<p>You can declare many variables in one statement.</p>
<p id="demo"></p>
<script>
var person = "John Doe";
var carName = "Volvo";
var price = 200;
document.getElementById("demo").innerHTML = person +"  
" + carName + "  
" + price;
</script>
</body>
</html>
```



# *Important Questions?*

- What is difference between undefined and null in java script?
- Explain primitive data type in java script.



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-III**

## ***Java Script: Statement***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

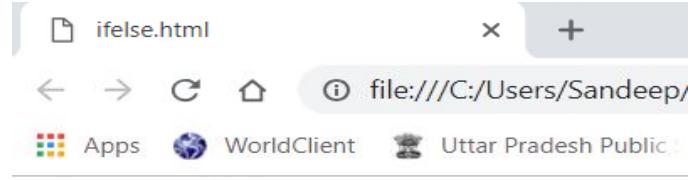
- **if...else Statement**
- **Switch Case**
- **While Loops**
- **For Loop**
- **Loop Control**
- **Important Questions**

# if...else Statement

- JavaScript supports conditional statements which are used to perform different actions based on different conditions.

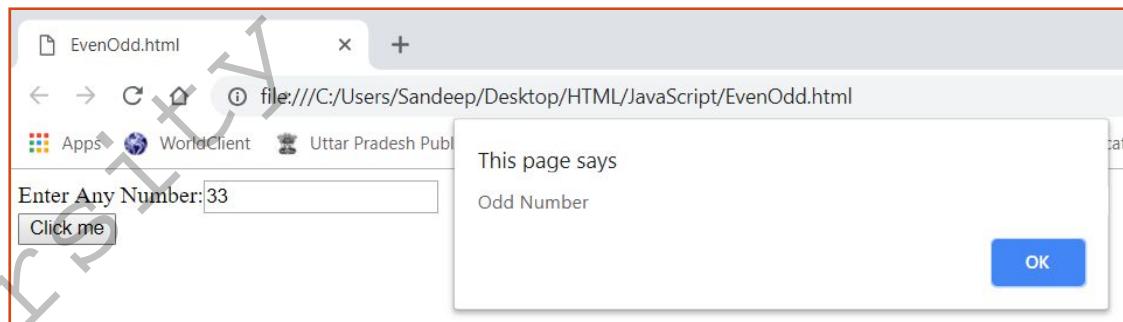
```
<html>
  <body>
    <script type="text/javascript">
      var age = 15;

      if( age > 18 ){
        document.write("Qualifies for driving");
      }
      else{
        document.write("Does not qualify for driving");
      }
    </script>
  </body>
</html>
```



# Example

```
<!doctype html>
<html>
<head>
<script>
function odd_even(){
var no;
no=Number(document.getElementById("no_input").value);
if(no%2==0)
{
alert("Even Number");
}
else
{
alert("Odd Number");
}
}
</script>
</head>
<body>
Enter Any Number:<input id="no_input"><br />
<button onclick="odd_even()">Click me</button>
</body>
</html>
```



# Switch Case

```
<!DOCTYPE html>
<html>
<body>
<H1 id="demo"></p>
<script>
var day;
switch (new Date().getDay()) {
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
```

case 4:

```
    day = "Thursday";
    break;
```

case 5:

```
    day = "Friday";
    break;
```

case 6:

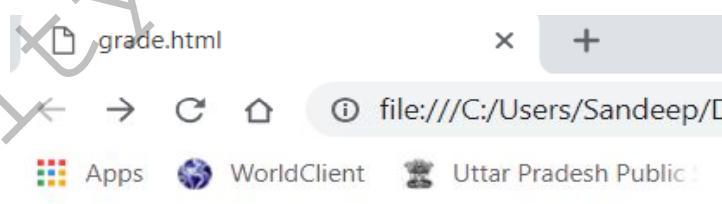
```
    day = "Saturday";
}
```

```
document.getElementById("demo").innerHTML = "Today is " + day;
```

```
</script>
```

```
</body>
```

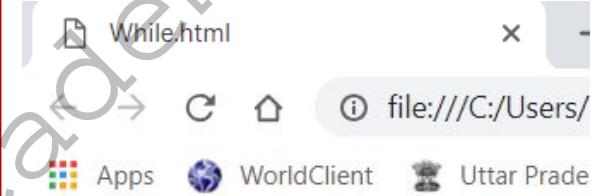
```
</html>
```



**Today is Monday**

# While Loops

```
<html>
<body>
<script type="text/javascript">
var count = 0;
document.write("Starting Loop ");
while (count < 10){
    document.write("Current Count : " + count + "<br />");
    count++;
}
document.write("Loop stopped!");
</script>
</body>
</html>
```



```
Starting Loop Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

# For Loop

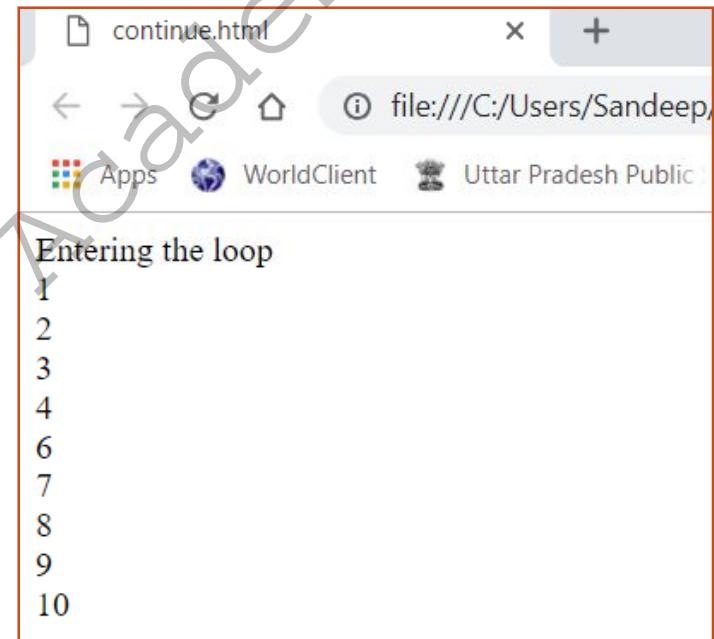
```
<html>
<body>
<script type="text/javascript">
var count ;
document.write("Starting Loop ");
for (count =0; count< 10; count++){
document.write("Current Count : " + count + "<br />");
}
document.write("Loop stopped!");
</script>
</body>
</html>
```



# Loop Control

- **break and continue**

```
<html>
<body>
<script type="text/javascript">
    var x = 0;
    document.write("Entering the loop<br />");
    while (x < 10)
    {
        x = x + 1;
        if (x == 5){
            continue; // skip rest of the loop body
        }
        document.write( x + "<br />");
    }
</script>
</body>
</html>
```



# Important Questions

1. Write a java script program to find largest of among 5 number.
2. Develop a HTML and java script document to evaluate yhe root of quadratic equation.
3. Write a java script to built up clock.



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-III**

## ***Java Script: Function & Form***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- **Functions**
- **Form**
- **Important Questions**

# Function

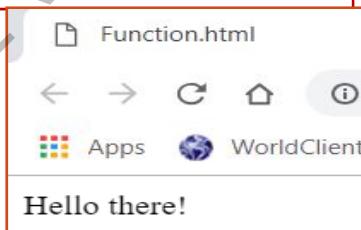
- A function is a group of reusable code which can be called anywhere in your program.
- Functions allow a programmer to divide a big program into a number of small and manageable functions.
- JavaScript also supports all the features necessary to write modular code using functions.
- You must have seen functions like **alert()** and **write()** in the earlier
- JavaScript allows us to write our own functions as well.

# Function Definition

- Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword.
- *Syntax*

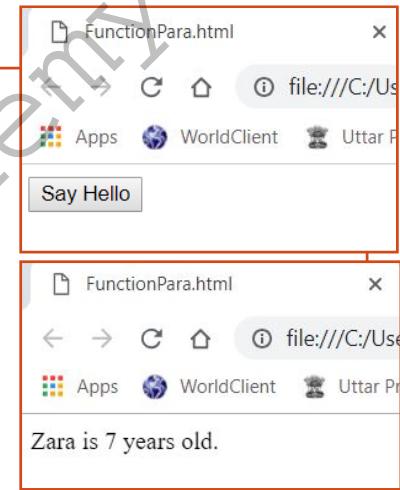
```
<script type="text/javascript">
    function functionname(parameter-list)
    {
        statements
    }
</script>
```

```
<html>
<head>
<script type="text/javascript">
function sayHello() {
document.write ("Hello there!");
}
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello">
</body>
</html>
```



# Function Parameters

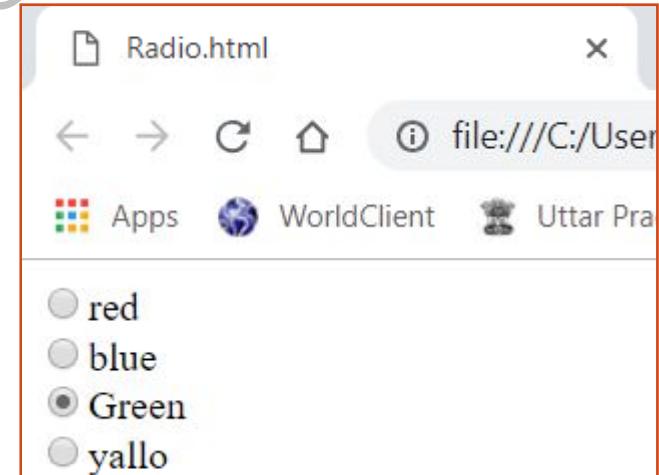
```
<html>
<head>
    <script type="text/javascript">
        function sayHello(name, age)
        {
            document.write (name + " is " + age + " years old.");
        }
    </script>
</head>
<body>
    <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</body>
</html>
```



# Forms

- Java scripting form is useful for designing interactive web page. Various control can be placed on this form are: Buttons, Radio buttons, checkbox, text box dropdown menu and so on.
- Radio Buttons

```
<html>
<head>
<script type="text/javascript">
function fun(form_obj)
{
    alert("Your favourite color is " + form_obj.value)
}
</script>
</head>
<body>
<input type="radio" value="red" onclick=fun(this)>red<br/>
<input type="radio" value="blue" onclick=fun(this)>blue<br/>
<input type="radio" value="green" onclick=fun(this)>Green<br/>
<input type="radio" value="yellow" onclick=fun(this)>yallo<br/>
</body>
</html>
```



# Forms

- Validation: we have used password verification process that come along the web document.

```
<html>
<head>
<script type="text/javascript">
function my_fun(form_obj)
{
    var mypwd= document.getElementById("pwd");
    var my_re_pwd= document.getElementById("re_pwd");
    if(mypwd.value=="")
    {
        alert("You Have not Entered the password");
    }
    if(mypwd.value!=my_re_pwd.value)
    {
        alert("password is not verified. Re enter");
    }
    else
        alert("Congratulations");
}
</script>
</head>
```

```
<body>
<center>
<form id="form1">

    <label>Enter Your Password <input type="password" value="" id="pwd"/></label><br/><br/>
    <label>Re Enter Your Password <input type="password" value="" id="re_pwd" onblur="my_fun();"/></label><br/>
    <input type="submit" value="Submit" name="submit" onsubmit="my_fun();"/>
</form>
</center>
</body>
</html>
```

# Important Questions

- Write java script function for email address validation
- Write java script function to validate user name.
- Write java script function for valid name and password field more than 6 character and age field range for 1-99 year.
- Briefly explain the form validation using JavaScript.



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-III**

## ***Java Script: Object***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- Introduction to Object
- Array Object
- String Object
- Number Object
- Boolean Object
- Date Object
- Math Object

# Introduction to Object

- A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.
- JavaScript is an object-based language. Everything is an object in JavaScript.
- JavaScript is template based not class based.
- Here, we don't create class to get the object. But, we directly create objects.

# Introduction to Object

- There are 3 ways to create objects.
  - By object literal
  - By creating instance of Object directly (using new keyword)
  - By using an object constructor (using this keyword)
- ***JavaScript Object by object literal*** : the syntax is

```
object={property1:value1,property2:value2.....propertyN:valueN}
```

- ***Example:***

```
<html>
<body>
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+
"+emp.salary);
</script>
</body>
</html>
```

Output:  
102 Shyam Kumar 40000

# Introduction to Object

- **By creating instance of Object:** Here, new keyword is used to create object. The syntax is

```
var objectname=new Object();
```

- **Example**

```
<html>
<body>
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```

Output:  
101 Ravi Malik 50000

# Introduction to Object

- By using an Object constructor: Here, you need to create function with arguments. Each argument value can be assigned in the current object by using **this** keyword.
- The **this keyword** refers to the current object.

```
<html>
<body>
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
</body>
</html>
```

Output:  
103 Vimal Jaiswal 30000

# Array Object

- The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type.
- **Syntax:** Use the following syntax to create an **Array** object –

```
var fruits = new Array( "apple", "orange", "mango" );
```

Or

```
var fruits = [ "apple", "orange", "mango" ];
```

- The **Array** parameter is a list of strings or integers. The maximum length allowed for an array is 4,294,967,295.
- You will use ordinal numbers to access and to set values inside an array as follows.

fruits[0] is the first element

fruits[1] is the second element

fruits[2] is the third element

## JavaScript [arrobj.html]

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<title> Array Object</title>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
a=new Array(10,20,30,40);
```

```
document.write("The original Array is : "+a);
```

```
a.push(50);
```

```
document.write("<br/>The Array after push method is :");
```

```
document.write(a);
```

```
a.pop();
```

```
document.write("<br/>The Array after pop method is :");
```

```
document.write(a);
```

```
a.reverse();
```

```
document.write("<br/>The Array after reverse method is :");
```

```
document.write(a);
```

```
a.shift();
```

```
document.write("<br/>The Array after shift method is :");
```

```
document.write(a);
```

```
a.sort();
```

```
document.write("<br/>The Array after sort method is :");
```

```
document.write(a);
```

```
a.push(11);
```

```
document.write("<br/>The Array after push method is :");
```

```
document.write(a);
```

```
a.splice(3,0,31);
```

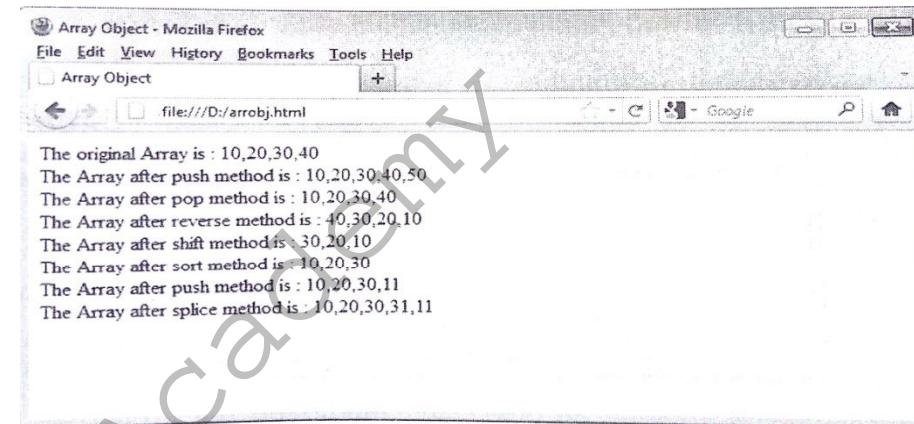
```
document.write("<br/>The Array after splice method is :");
```

```
document.write(a);
```

```
</script>
```

```
</body>
```

```
</html>
```



# String Object

- The **JavaScript string** is an object that represents a sequence of characters.
- Some commonly used method of string objects are concatenation, converting to upper or lower case, length etc.
- There are 2 ways to create string in JavaScript
  - By string literal
  - By string object (using new keyword)

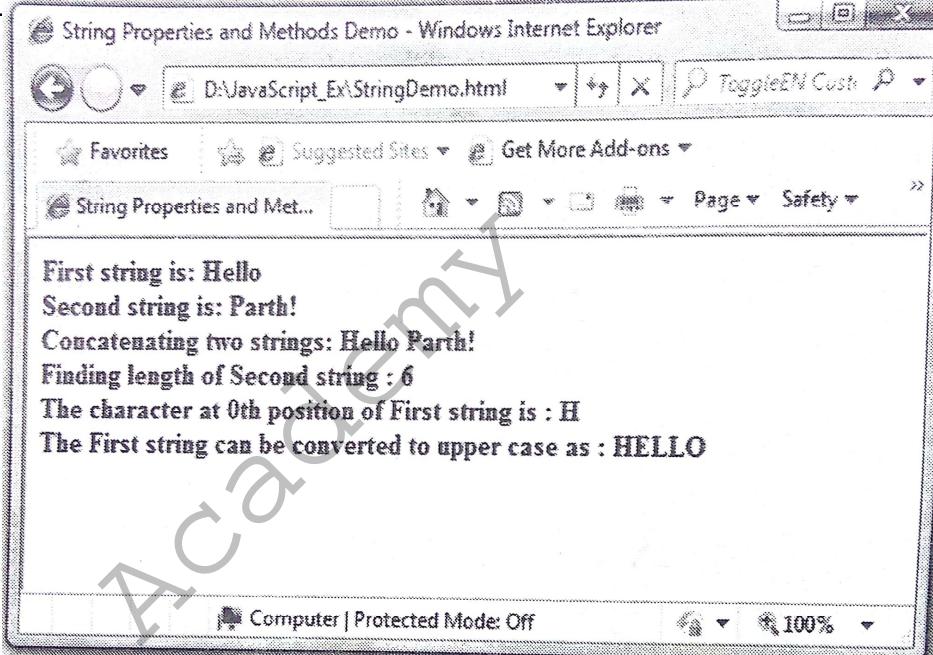
```
var stringname="string value";
```

```
var stringname=new String("string literal");
```

```

<head>
  <title>String Properties and Methods Demo</title>
</head>
<body>
<strong>
  <script type="text/javascript">
    var s1="Hello ";
    var s2="Parth!";
    document.write("First string is: "+s1+"<br>");
    document.write("Second string is: "+s2+"<br>");
    document.write("Concatenating two strings: "+s1.concat(s2)+"<br>");
    document.write("Finding length of Second string : "+s2.length+"<br>");
    document.write("The character at 0th position of First string is
      "+s1.charAt(0)+"<br>");
    document.write("The First string can be converted to upper case as
      "+s1.toUpperCase());
  </script>
</strong>
</body>
</html>

```



# JavaScript Number Object

- The **JavaScript number** object *enables you to represent a numeric value.*
- It may be integer or floating-point.
- JavaScript number object follows IEEE standard to represent the floating-point numbers.
- By the help of Number() constructor, you can create number object in JavaScript.
- For example:

```
var n=new Number(value);
```

# Example

```
<!DOCTYPE html>
<html>
<body>
<script>
var x=102;//integer value
var y=102.7;//floating point value
var z=13e4;//exponent value, output: 130000
var n=new Number(16);//integer value by number object
document.write(x+" "+y+" "+z+" "+n);
</script>
</body>
</html>
```

Output:  
102 102.7 130000 16

# Number Properties

Sr.No	Property & Description
1	<b><u>MAX_VALUE</u></b> The largest possible value a number in JavaScript can have 1.7976931348623157E+308
2	<b><u>MIN_VALUE</u></b> The smallest possible value a number in JavaScript can have 5E-324
3	<b><u>NaN</u></b> Equal to a value that is not a number.
4	<b><u>NEGATIVE_INFINITY</u></b> A value that is less than MIN_VALUE.
5	<b><u>POSITIVE_INFINITY</u></b> A value that is greater than MAX_VALUE
6	<b><u>Prototype</u></b> A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document
7	<b><u>Constructor</u></b> Returns the function that created this object's instance. By default this is the Number object.

# Number Methods

Sr.No	Method & Description
1	<b><u>toExponential()</u></b> Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
2	<b><u>toFixed()</u></b> Formats a number with a specific number of digits to the right of the decimal.
3	<b><u>toLocaleString()</u></b> Returns a string value version of the current number in a format that may vary according to a browser's local settings.
4	<b><u>toPrecision()</u></b> Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
5	<b><u>toString()</u></b> Returns the string representation of the number's value.
6	<b><u>valueOf()</u></b> Returns the number's value.

# Example

```
<html>
<head>
<title>Javascript Method toExponential()</title>
</head>
<body>
<script type="text/javascript">
var num=77.1234;
var val = num.toExponential();
document.write("num.toExponential() is : " + val );
document.write("<br />");
val = num.toExponential(4);
document.write("num.toExponential(4) is : " + val );
document.write("<br />");
val = num.toExponential(2);
document.write("num.toExponential(2) is : " + val );
document.write("<br />"); val = 77.1234.toExponential();
document.write("77.1234.toExponential()is : " + val );
document.write("<br />");
val = 77.1234.toExponential();
document.write("77 .toExponential() is : " + val );
</script>
</body>
</html>universityacademy.in
```

num.toExponential() is : 7.71234e+1  
num.toExponential(4) is : 7.7123e+1  
num.toExponential(2) is : 7.71e+1  
77.1234.toExponential()is:7.71234e+1  
77 .toExponential() is : 7.71234e+1

# JavaScript - The Boolean Object

- The **Boolean** object represents two values, either "true" or "false".
- If *value* parameter is omitted or is 0, -0, null, false, **NaN**, undefined, or the empty string (""), the object has an initial value of false.
- You can create the JavaScript Boolean object by Boolean() constructor as given below.

```
var val = new Boolean(value);
```

# Boolean Methods

Sr.No	Method & Description
1	<b><u>toSource()</u></b> Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	<b><u>toString()</u></b> Returns a string of either "true" or "false" depending upon the value of the object.
3	<b><u>valueOf()</u></b> Returns the primitive value of the Boolean object

# Example

```
<html>
<head>
<title>JavaScript toSource() Method</title>
</head>
<body>
<script type="text/javascript">
function book(title, publisher, price)
{
this.title = title;
this.publisher = publisher;
this.price = price;
}
var newBook = new book("Perl", "Leo Inc", 200);
document.write(newBook.toSource());
</script>
</body>
</html>
```

({title:"Perl", publisher:"Leo Inc", price:200})

# Date Object

- The **JavaScript date** object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.
- You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.
  - Date()
  - Date(milliseconds)
  - Date(dateString)
  - Date(year, month, day, hours, minutes, seconds, milliseconds)

# JavaScript Date Methods

Method	Description
<a href="#"><u>getDate()</u></a>	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.
<a href="#"><u>getDay()</u></a>	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.
<a href="#"><u>getFullYears()</u></a>	It returns the integer value that represents the year on the basis of local time.
<a href="#"><u>getHours()</u></a>	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.
<a href="#"><u>getMilliseconds()</u></a>	It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.
<a href="#"><u>getMinutes()</u></a>	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.
<a href="#"><u>getMonth()</u></a>	It returns the integer value between 0 and 11 that represents the month on the basis of local time.
<a href="#"><u>getSeconds()</u></a>	It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.
<a href="#"><u>getUTCDate()</u></a>	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of universal time.
<a href="#"><u>getUTCDay()</u></a>	It returns the integer value between 0 and 6 that represents the day of the week on the basis of universal time.

# Example

```
<html>
<body>
Current Time: <span id="txt"></span>
<script>
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
document.getElementById('txt').innerHTML=h+":"+m+":"+s;
</script>
</body>
</html>
```

Output:  
Current Time: 9:10:11

# Example

```
<html>
  <head>
<title>JavaScript getDate Method</title>
</head>
<body>
<script type="text/javascript">
  var dt = new Date();
  document.write("getDate() : " + dt.getDate() );
</script>
</body>
</html>
```

Output:  
getDate() : 24

# Example

```
<html>
<head>
<title>JavaScript getDate Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date("December 24, 1995 23:15:00");
document.write("get Date : " + dt.getDate());
document.write("<br>getYear : " + dt.getYear());
document.write("<br>getTime: " + dt.getTime());
document.write("<br>get Date: " + dt.getDate());
document.write("<br>get Hours: " + dt.getHours());
document.write("<br>get Minutes: " + dt.getMinutes());
</script>
</body>
</html>
```

Output:  
get Date : 24  
getYear : 95  
getTime:  
819827100000  
get Date: 24  
get Hours: 23  
get Minutes: 15

# JavaScript Math

- The **math** object provides you properties and methods for mathematical constants and functions.
- Unlike other global objects, **Math** is not a constructor.
- All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.
- Thus, you refer to the constant **pi** as **Math.PI** and you call the *sine* function as **Math.sin(x)**, where x is the method's argument.

```
var pi_val = Math.PI;  
var sine_val = Math.sin(30);
```

# JavaScript Math Methods

Methods	Description
<a href="#">abs()</a>	It returns the absolute value of the given number.
<a href="#">acos()</a>	It returns the arccosine of the given number in radians.
<a href="#">asin()</a>	It returns the arcsine of the given number in radians.
<a href="#">atan()</a>	It returns the arc-tangent of the given number in radians.
<a href="#">cbrt()</a>	It returns the cube root of the given number.
<a href="#">ceil()</a>	It returns a smallest integer value, greater than or equal to the given number.
<a href="#">cos()</a>	It returns the cosine of the given number.
<a href="#">cosh()</a>	It returns the hyperbolic cosine of the given number.
<a href="#">exp()</a>	It returns the exponential form of the given number.
<a href="#">floor()</a>	It returns largest integer value, lower than or equal to the given number.
<a href="#">hypot()</a>	It returns square root of sum of the squares of given numbers.
<a href="#">log()</a>	It returns natural logarithm of a number.
<a href="#">max()</a>	It returns maximum value of the given numbers.
<a href="#">min()</a>	It returns minimum value of the given numbers.
<a href="#">pow()</a>	It returns value of base to the power of exponent.
<a href="#">random()</a>	It returns random number between 0 (inclusive) and 1 (exclusive).
<a href="#">round()</a>	It returns closest integer value of the given number.
<a href="#">sign()</a>	It returns the sign of the given number
<a href="#">sin()</a>	It returns the sine of the given number.
<a href="#">sinh()</a>	It returns the hyperbolic sine of the given number.
<a href="#">sqrt()</a>	It returns the square root of the given number
<a href="#">tan()</a>	It returns the tangent of the given number.
<a href="#">tanh()</a>	It returns the hyperbolic tangent of the given number.
<a href="#">trunc()</a>	It returns an integer part of the given number.

# Example

```
<html>
<head>
<title>JavaScript Math abs() Method</title>
</head>
<body>
<script type="text/javascript">
var value = Math.abs(-1);
document.write("First Test Value : " + value);
var value = Math.abs(null);
document.write("<br />Second Test Value : " + value );
var value = Math.abs(20);
document.write("<br />Third Test Value : " + value );
var value = Math.abs("string");
document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output  
First Test Value : 1  
Second Test Value : 0  
Third Test Value : 20  
Fourth Test Value : NaN

# Example

```
<html>
<head>
<title>JavaScript Math sqrt() Method</title>
</head>
<body>
<script type="text/javascript">
var value = Math.sqrt( 0.5 );
document.write("First Test Value : " + value );
var value = Math.sqrt( 81 );
document.write("<br />Second Test Value : " + value );
var value = Math.sqrt( 13 );
document.write("<br />Third Test Value : " + value );
var value = Math.sqrt( -4 );
document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

## Output

First Test Value : 0.7071067811865476  
Second Test Value : 9  
Third Test Value : 3.605551275463989  
Fourth Test Value : NaN

# Important Questions

- Develop a HTML and Javascript document to root of quadratic equation.
- Explain any three object that can be defined in java script with help of example.
- Write a java script to built up clock.



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-III**

## ***Ajax,DHTML,DOM***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

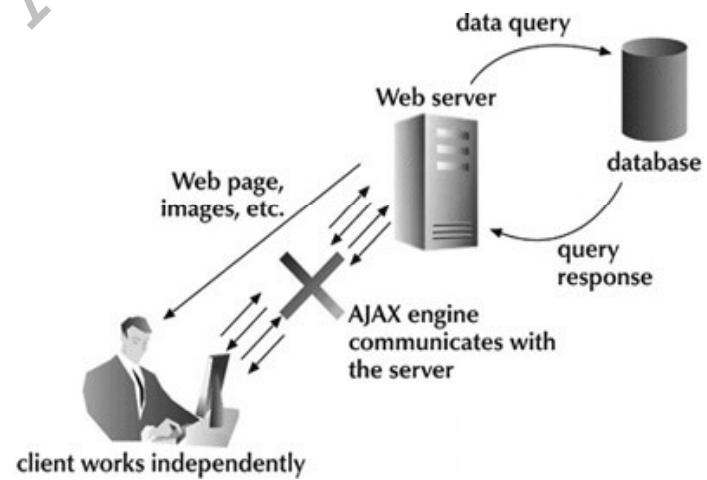
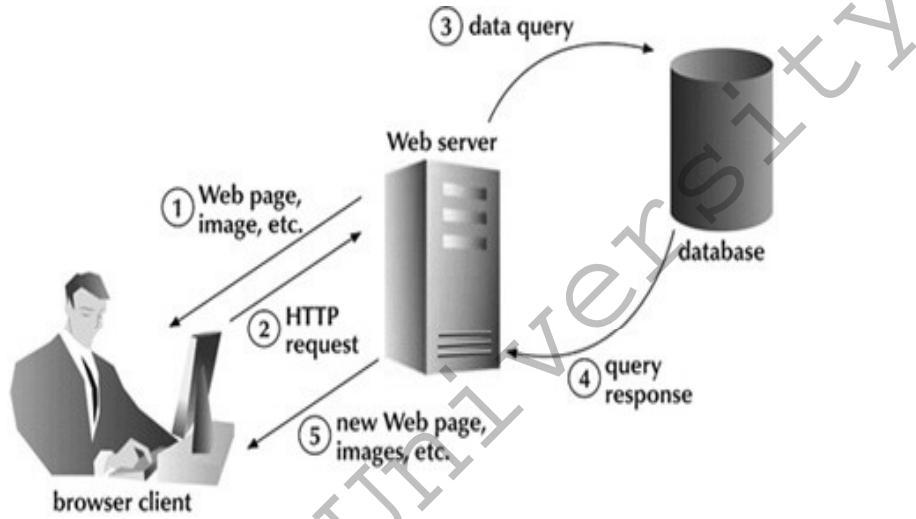
- What is AJAX?
- AJAX - Technologies
- How AJAX Works
- AJAX – Examples
- DHTML
- DOM

# What is AJAX?

- AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like XML, HTML, CSS, and Java Script.etc.
- AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.
- AJAX allows you to send only important information to the server not the entire page
- So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.
- There are too many web applications running on the web that are using ajax technology like **gmail, facebook, twitter, google map, youtube** etc.

# Understanding Synchronous vs Asynchronous

- A synchronous request blocks the client until operation completes i.e. browser is unresponsive. In such case, javascript engine of the browser is blocked.
- An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.



# AJAX - Technologies

- As describe earlier, ajax is not a technology but group of inter-related technologies. AJAX technologies includes:
- HTML/XHTML and CSS : These technologies are used for displaying content and style. It is mainly used for presentation
- DOM:It is used for dynamic display and interaction with data.
- XML or JSON: For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.
- XMLHttpRequest: For asynchronous communication between client and server.
- JavaScript: It is used to bring above technologies together.

# XMLHttpRequest

- An object of XMLHttpRequest is used for asynchronous communication between client and server.
  - It performs following operations:
  - Sends data from the client in the background
  - Receives the data from the server
  - Updates the webpage without reloading it.

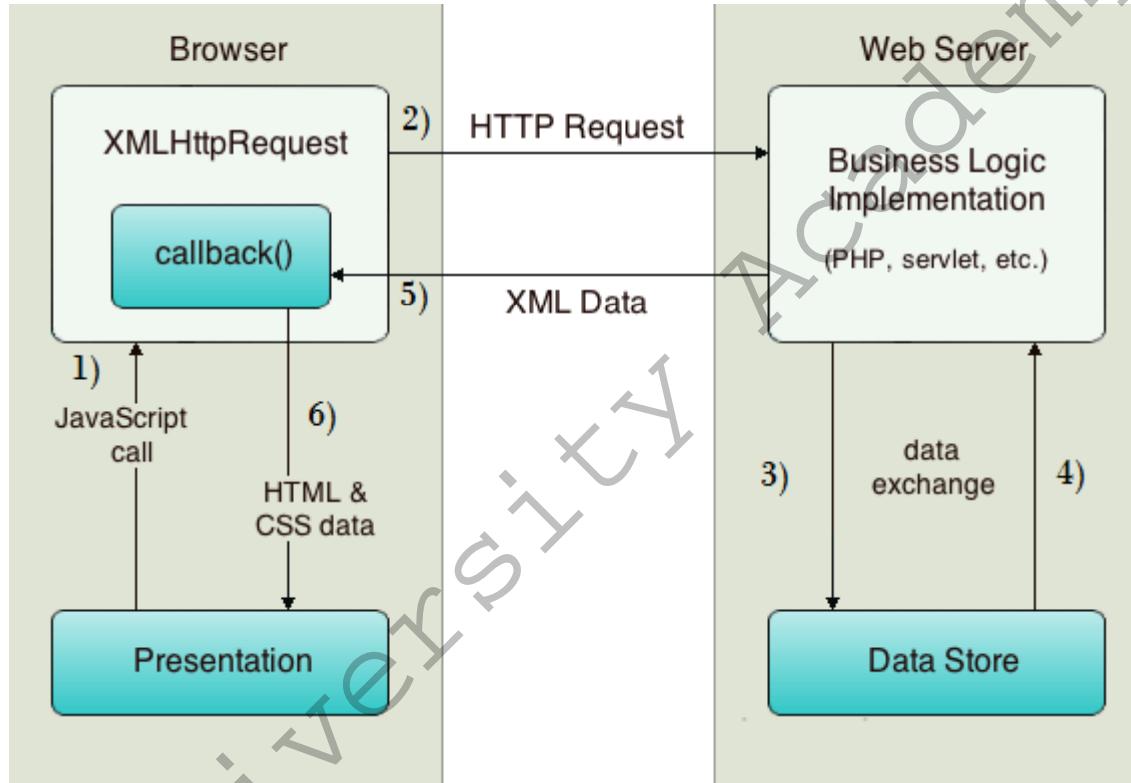
# Properties of XMLHttpRequest object

Property	Description
onReadyStateChange	It is called whenever readyState attribute changes. It must not be used with synchronous requests.
readyState	represents the state of the request. It ranges from 0 to 4. <b>0 UNOPENED</b> open() is not called. <b>1 OPENED</b> open is called but send() is not called. <b>2 HEADERS_RECEIVED</b> send() is called, and headers and status are available. <b>3 LOADING</b> Downloading data; responseText holds the data. <b>4 DONE</b> The operation is completed fully.
reponseText	returns response as text.
responseXML	returns response as XML

# Methods of XMLHttpRequest object

Method	Description
void open(method, URL)	opens the request specifying get or post method and url.
void open(method, URL, async)	same as above but specifies asynchronous or not.
void open(method, URL, async, username, password)	same as above but specifies username and password.
void send()	sends get request.
void send(string)	send post request.
setRequestHeader(header,value)	it adds request headers.

# How AJAX works?



# Ajax Java Example

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>

<script>
function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML =
            this.responseText;
        }
    };
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
}
</script>

</body>
</html>
```

## The XMLHttpRequest Object

Change Content

## AJAX

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.

# DHTML

- DHTML stands for **Dynamic HTML**. DHTML is NOT a language or a web standard.
- DHTML is a TERM used to describe the technologies used to make web pages dynamic and interactive.
- To most people DHTML means the combination of HTML, JavaScript and CSS.

According to the World Wide Web Consortium (W3C):

*"Dynamic HTML is a term used by some vendors to describe the combination of HTML, style sheets and scripts that allows documents to be animated."*

# HTML Vs DHTML

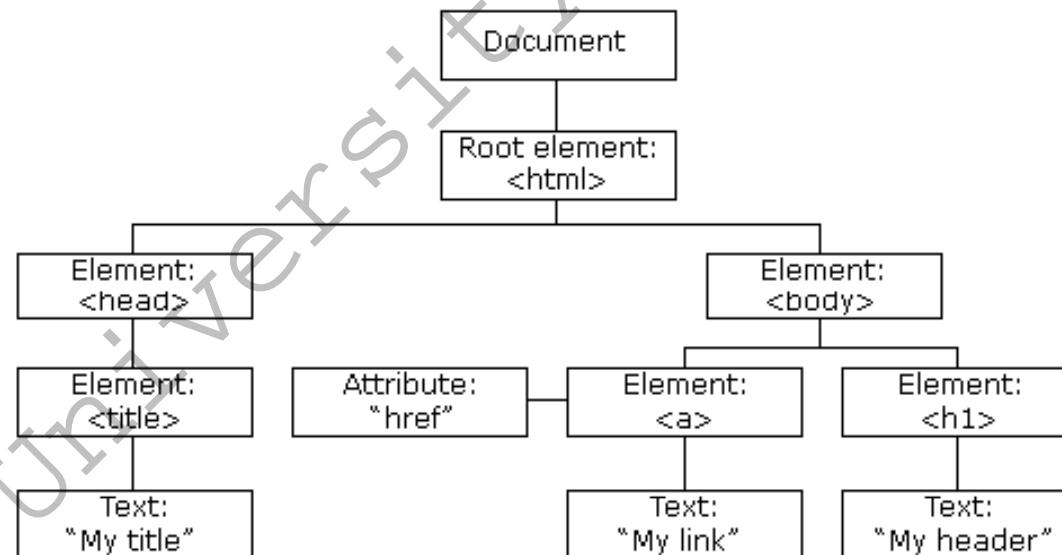
- HTML is a mark-up language, while DHTML is a collection of technology.
- DHTML creates dynamic web pages, whereas HTML creates static web pages.
- DHTML allows including small animations and dynamic menus in Web pages.
- DHML used events, methods, properties to insulate dynamism in HTML Pages.
- DHML is basically using JavaScript and style sheets in an HTML page.
- HTML sites will be slow upon client-side technologies, while DHTML sites will be fast enough upon client-side technologies.
- HTML creates a plain page without any styles and Scripts called as HTML. Whereas, DHTML creates a page with HTML, CSS, DOM and Scripts called as DHTML.
- HTML cannot have any server side code but DHTML may contain server side code.
- In HTML, there is no need for database connectivity, but DHTML may require connecting to a database as it interacts with user.
- HTML files are stored with .htm or .html extension, while DHTML files are stored with .dhtm extension.
- HTML does not require any processing from browser, while DHTML requires processing from browser which changes its look and feel.

# DOM

- The Document Object Model (DOM) is a programming API for HTML and XML documents.
- It defines the logical structure of documents and the way a document is accessed and manipulated.
- As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications.
- The Document Object Model can be used with any programming language.

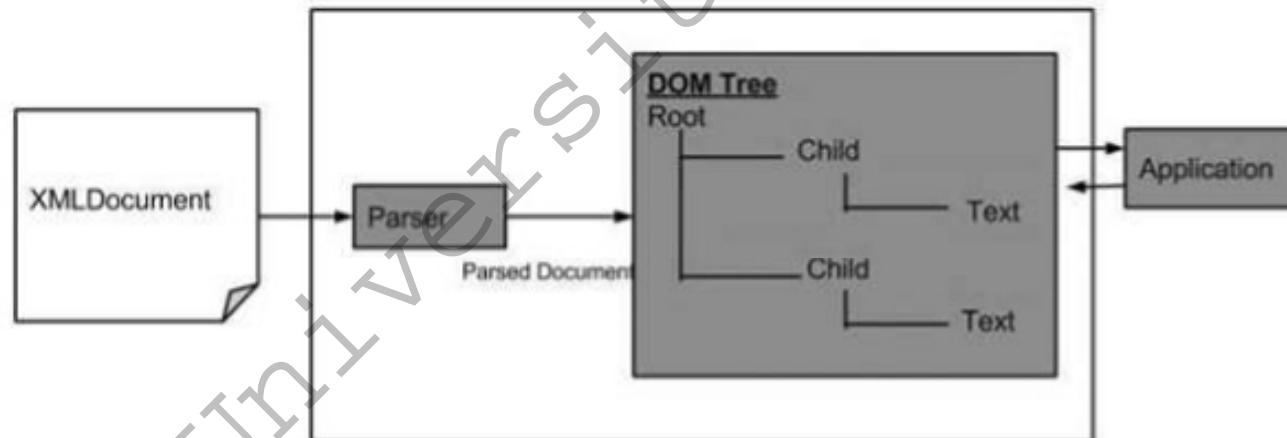
# HTML DOM

- The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:
  - The HTML elements as **objects**
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements



# XML DOM

- XML DOM is a standard object model for XML.
- XML documents have a hierarchy of informational units called *nodes*;
- DOM is a standard programming interface of describing those nodes and the relationships between them.



# Important Question?

- What are DHTML? How DHTML is used with JavaScript?
- Differentiate between HTML and DHTML.
- Write Short Note on AJAX.
- What do you mean by AJAX ? Explain the Application of AJAX with example.
- Write Short notes on AJAX.
- What do you mean by DOM? Design DOM for Student.



**University Academy**  
Teaching|Training|Informative

# Web Technology (Java programming)

**Unit-IV**

## *Enterprise Java Beans(EJB)*

By

Mr. Sandeep Vishwakarma

Assistant Professor

Dr. A.P.J. Abdul Kalam

Technical University,Lucknow

# Summary

- Introduction
- Architecture
- Types of EJB
- Important Questions?

# Introduction

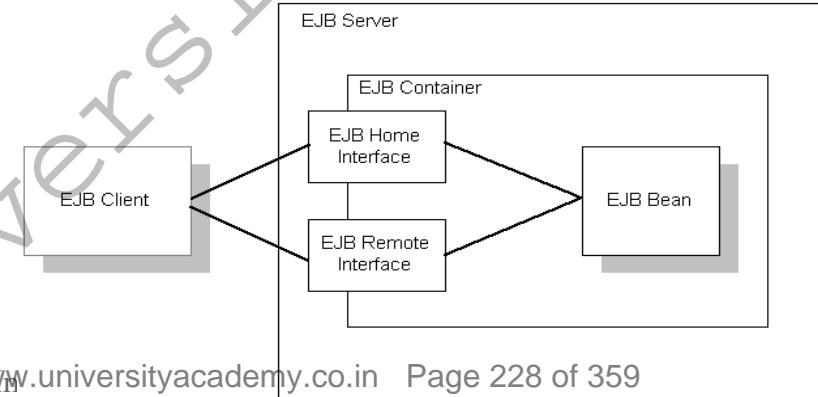
- EJB stands for **Enterprise Java Beans**.
- Enterprise Java Beans is a server side component.
- EJB is an essential part of a J2EE platform
- J2EE application container contains the components that can be used by client for executing business logic. These components are called EJB.
- EJB Mainly contain the business logic and business data.
- EJB component always lie in some container which is called EJB container.
- EJB component is an EJB class which is written by developer that implement business logic.

# Introduction

- It is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications.
- To run EJB application, you need an *application server* (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc.
- EJB application is deployed on the server, so it is called server side component also.
- EJB Container performs:
  - life cycle management,
  - security,
  - transaction management, and
  - object pooling.

# EJB Architecture

- **EJB server:** The EJB server contains the EJB container
- **EJB client:** An EJB client usually provides the user-interface logic on a client machine
- An EJB client does not communicate directly with an EJB component
  - **remote interface** defines the business methods that can be called by the client.
  - **home interface** methods to create and destroy proxies for the remote interface
- **EJB container** The EJB specification defines a container as the environment in which one or more EJB components execute.

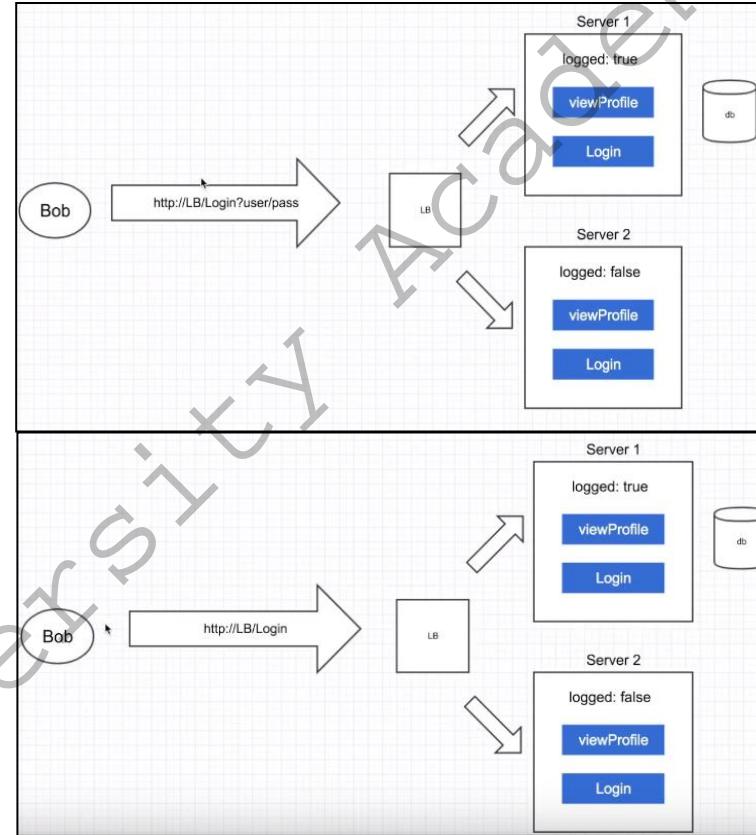


# Types of EJB

- EJB is primarily divided into three categories;
- Session Bean
  - Stateful
  - Stateless
- Entity Bean
- Message Driven Bean

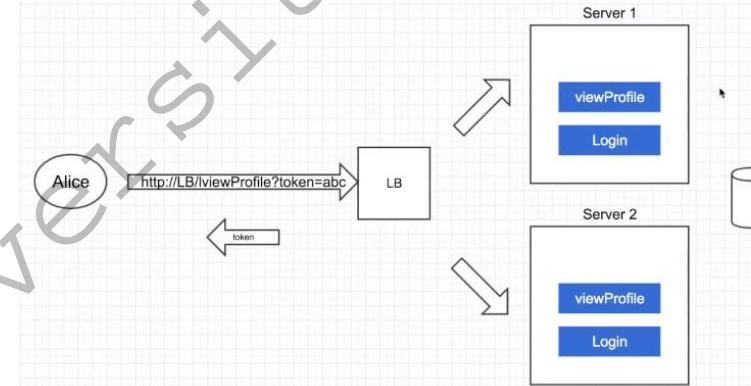
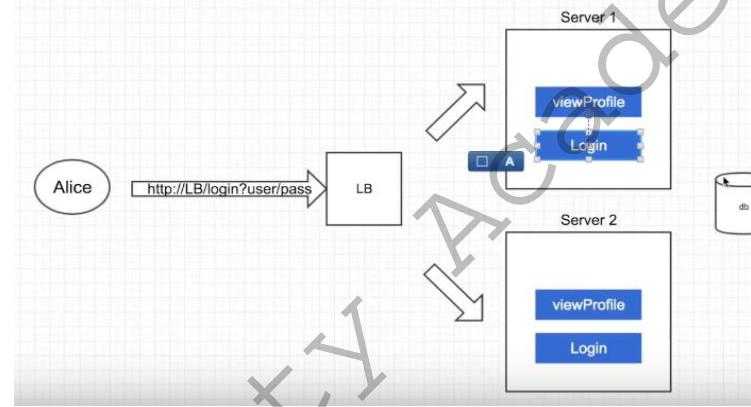
# stateful

**Stateful Session Bean:** It maintains state of a client across multiple requests.



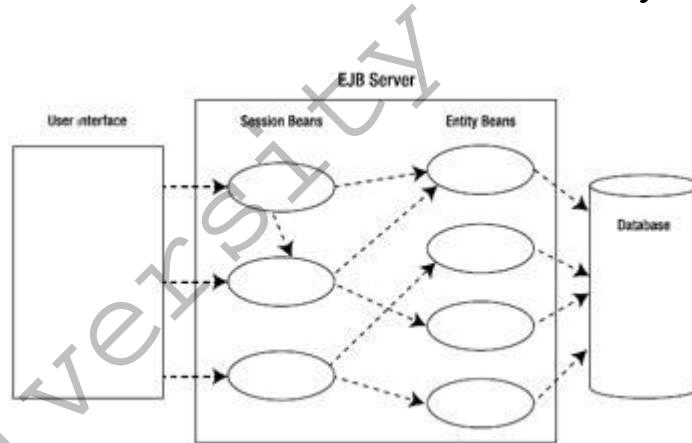
# Stateless

**Stateless Session Bean:** It doesn't maintain state of a client between multiple method calls.



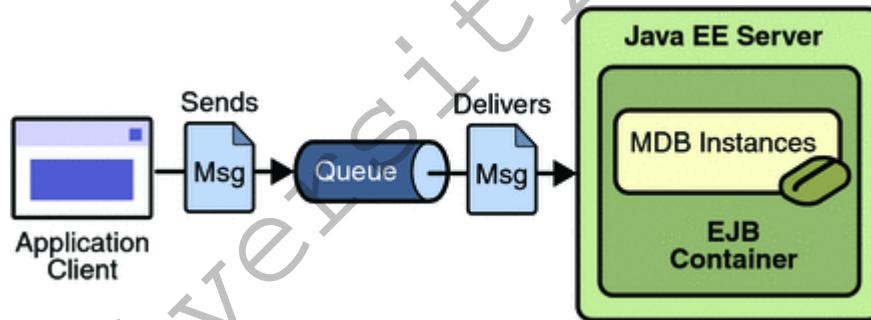
# Entity Bean

- Entity Bean represent persistent data storage. User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean.



# Message Driven Bean

- **Message driven beans** are used in context of JMS (Java Messaging Service). Message Driven Beans can consumes JMS messages from external entities and act accordingly.



# Important Question

- How Many different types of EJB component are available? Explain.
- What is EJB? Differentiate Java Beans And EJB.
- Explain EJB and its architecture.
- Differentiate Stateful and Stateless EJB.



**University Academy**  
Teaching|Training|Informative

# Web Technology (Java programming)

**Unit-V**

## *Java Database Connectivity(JDBC)*

By

Mr. Sandeep Vishwakarma

Assistant Professor

Dr. A.P.J. Abdul Kalam

Technical University,Lucknow

# Summary

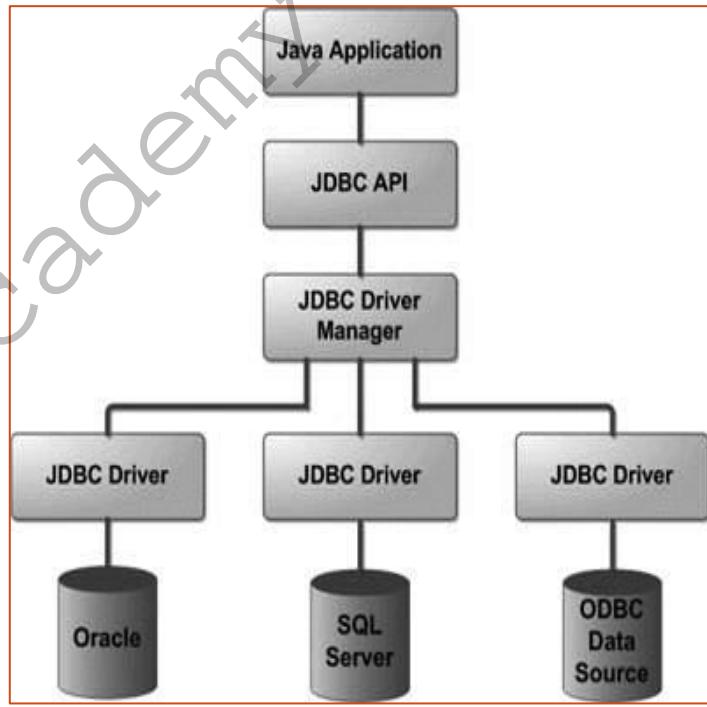
- JDBC Introduction
- JDBC Driver
- Databases with JDBC
- Prepared Statements
- Transaction Processing
- Stored Procedures
- Important Questions?

# JDBC Introduction

- JDBC stands for **Java Database Connectivity**, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases
- JDBC is used to interact with various type of Database such as Oracle, MS Access, MySQL and SQL Server.
- It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:
  - JDBC-ODBC Bridge Driver,
  - Native Driver,
  - Network Protocol Driver, and
  - Thin Driver

# JDBC Architecture

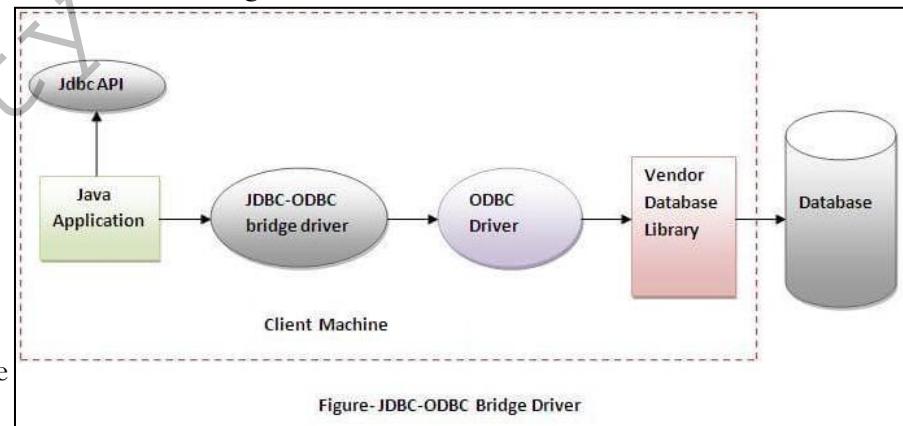
- The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers –
  - JDBC API:** This provides the application-to-JDBC Manager connection. The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
  - JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection. The JDBC driver manager ensures that the correct driver is used to access each data source.



Note: The **java.sql** package contains classes and interfaces for JDBC API

# JDBC-ODBC bridge driver

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.
- Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.
- Advantages:
  - easy to use.
  - can be easily connected to any database.
- Disadvantages:
  - Performance degraded because JDBC method call is converted into the ODBC function calls.
  - The ODBC driver needs to be installed on the client machine



# The Native API driver

- The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

## Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

## Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

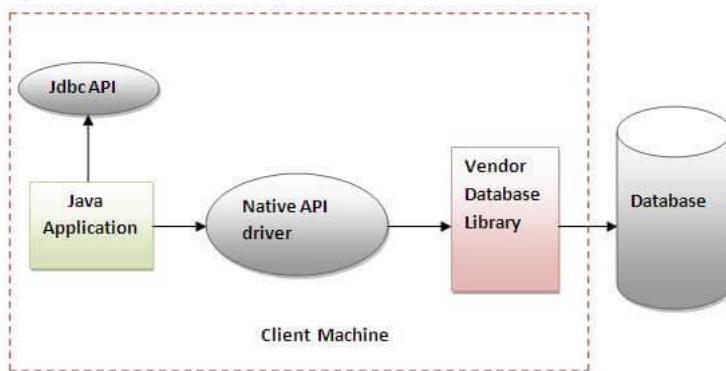


Figure- Native API Driver

# Network Protocol driver

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in Java:

Advantages:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

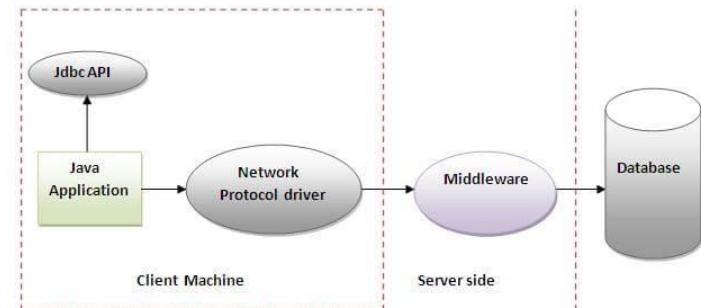


Figure- Network Protocol Driver

# Thin driver

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

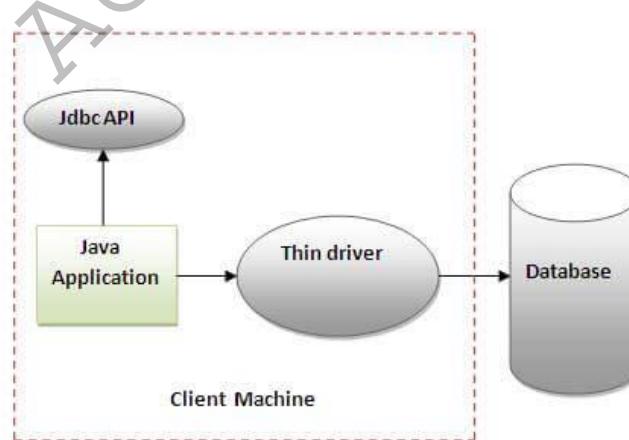


Figure-Thin Driver

# Databases with JDBC

- There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:
  - Register the Driver class
  - Create connection
  - Create statement
  - Execute queries
  - Close connection

Java Database Connectivity

Register driver

Get connection

Create statement

Execute query

Close connection



# Java Database Connectivity with Oracle

- **Create a Table:** Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

- To connect java application with the Oracle database ojdbc14.jar file is required to be loaded

```
import java.sql.*;  
class OracleCon{  
    public static void main(String args[]){  
        try{  
            //step1 load the driver class  
            Class.forName("oracle.jdbc.driver.OracleDriver");  
            //step2 create the connection object  
            Connection con=DriverManager.getConnection(  
                "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");  
            //step3 create the statement object  
            Statement stmt=con.createStatement();  
            //step4 execute query  
            ResultSet rs=stmt.executeQuery("select * from emp");  
            while(rs.next())  
                System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));  
            //step5 close the connection object  
            con.close();  
        } catch(Exception e){ System.out.println(e);}  
    }  
}
```

# Prepared Statement

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.
- example of parameterized query:

```
String sql="insert into emp values(?, ?, ?);
```

- As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

# Methods of PreparedStatement interface

- The important methods of PreparedStatement interface are given below:

Method	Description
public void setInt(int paramInt, int value)	sets the integer value to the given parameter index.
public void setString(int paramInt, String value)	sets the String value to the given parameter index.
public void setFloat(int paramInt, float value)	sets the float value to the given parameter index.
public void setDouble(int paramInt, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

## Example of PreparedStatement interface that inserts the record

First of all create table as given below:

```
create table emp(id number(10),name varchar2(50));
```

```
import java.sql.*;  
class InsertPrepared{  
    public static void main(String args[]){  
        try{  
            Class.forName("oracle.jdbc.driver.OracleDriver");  
            Connection con=DriverManager.getConnection(  
                "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");  
            PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");  
            stmt.setInt(1,101); // 1 specifies the first parameter in the query  
            stmt.setInt(2,"Ratan");  
            int i=stmt.executeUpdate();  
            System.out.println(i+" records inserted");  
            con.close();  
        } catch(Exception e) { System.out.println(e);}  
    }  
}
```

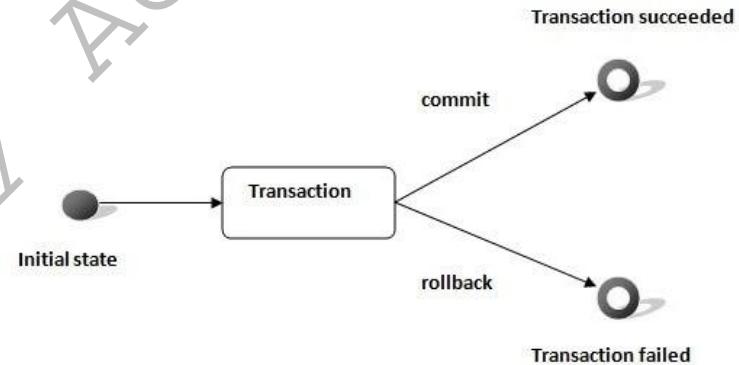
# Transaction Processing

- Transaction represents **a single unit of work**.
- The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.
- **Atomicity** means either all successful or none.
- **Consistency** ensures bringing the database from one consistent state to another consistent state.
- **Isolation** ensures that transaction is isolated from other transaction.
- **Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

# Transaction Processing

- In JDBC, **Connection interface** provides methods to manage transaction.

Method	Description
<code>void setAutoCommit(boolean status)</code>	It is true by default means each transaction is committed by default.
<code>void commit()</code>	commits the transaction.
<code>void rollback()</code>	cancels the transaction.



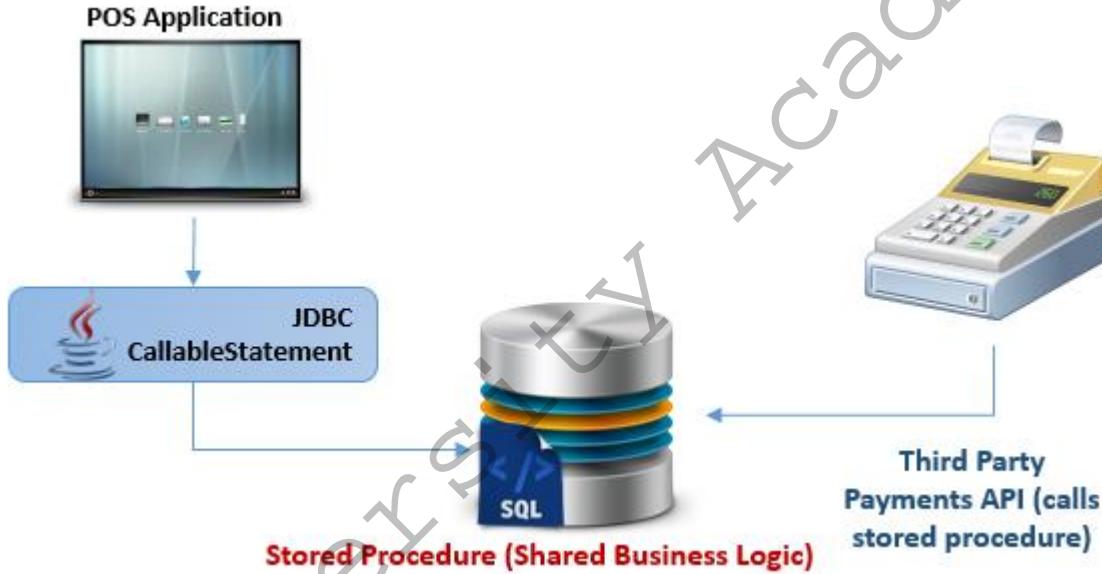
# Simple example of transaction

```
import java.sql.*;  
class FetchRecords{  
    public static void main(String args[])throws Exception{  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");  
        con.setAutoCommit(false);  
        Statement stmt=con.createStatement();  
        stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");  
        stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");  
        con.commit();  
        con.close();  
    }  
}
```

# Stored Procedures

- Stored procedures are a great way of incorporating business logic in the database.
- A stored procedure is basically a group of SQL statements put together that can be executed when the stored procedure is called.
- Let's say you have are developing a point of sale application. You have a desktop application that processes the payments and you also have some third party applications that need to process payments too.
- You can have the logic in your desktop application for posting payments and create a module responsible for handling the third party payments.
- Does the above sound like doing the same job twice? Yes it does. Stored procedures can help you in such cases

# Stored Procedures



# Important Questions?

- Explain JDBC Application Architecture.
- What are various type of JDBC drivers.
- Write steps to connect database with the web application using database.
- Explain javax.sql.\* package.
- What are statement in JDBC? Explain Prepared Statements.
- What are Transaction Processing. Explain ACID properties.
- Explain Stored Procedures with examples.



**University Academy**

Teaching|Training|Informative

# **Web Technology (Java programming)**

**Unit-III**

## ***Java Networking***

**By**

**Mr. Sandeep Vishwakarma**

**Assistant Professor**

**Dr. A.P.J. Abdul Kalam**

**Technical University,Lucknow**

# Summary

- Introduction?
- Java Socket Programming
- URL Class
- URLConnection class
- HttpURLConnection
- InetAddress class
- DatagramSocket class
- Important Questions?

# Introduction

- The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.
- The `java.net` package contains a collection of classes and interfaces that provide the communication details
- **Java Networking Terminology**
  - IP Address: eg: 192.168.0.1
  - Protocol : TCP,FTP,Telnet, SMTP, POP etc.
  - Port Number: The port number is associated with the IP.
  - MAC Address(Media Access Control)
  - Connection-oriented and connection-less protocol: Eg: TCP , UDP
  - Socket: A socket is an endpoint between two way communication.

# Socket Programming

- Sockets provide the communication mechanism between two computers.
- Java Socket programming can be connection-oriented or connection-less.
- A client program creates a socket on its end of the communication and attempts to connect that socket to a server.
- Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.
- The client in socket programming must know two information:
  - IP Address of Server, and
  - Port number.

# Socket class

- A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

# ServerSocket class

- The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

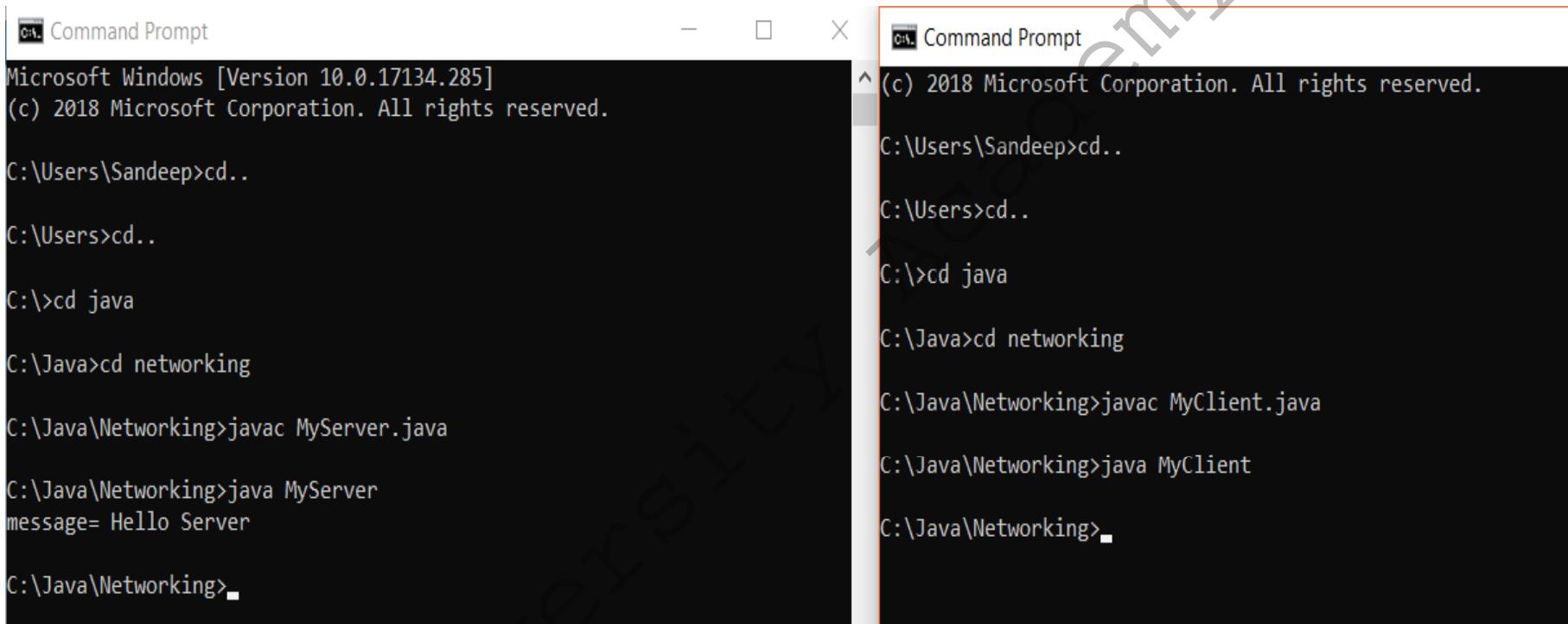
Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

# Example of Java Socket Programming

```
// MyServer.java
import java.io.*;
import java.net.*;
public class MyServer {
    public static void main(String[] args) {
        try{
            ServerSocket ss=new ServerSocket(6666);
            Socket s=ss.accept(); //establishes connection
            DataInputStream dis=new
            DataInputStream(s.getInputStream());
            String str=(String)dis.readUTF(); // Unicode
            Transformation Format
            System.out.println("message= "+str);
            ss.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

```
//MyClient.java
import java.io.*;
import java.net.*;
public class MyClient {
    public static void main(String[] args) {
        try{
            Socket s=new Socket("localhost",6666);
            DataOutputStream dout=new DataOutputStream(
            s.getOutputStream());
            dout.writeUTF("Hello Server");
            dout.flush();
            dout.close();
            s.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

# Output



The image shows two Microsoft Windows Command Prompt windows side-by-side. Both windows have a title bar labeled "Command Prompt". The window on the left has a dark background and displays the following command-line session:

```
Microsoft Windows [Version 10.0.17134.285]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Sandeep>cd..
C:\Users>cd..
C:\>cd java
C:\Java>cd networking
C:\Java\Networking>javac MyServer.java
C:\Java\Networking>java MyServer
message= Hello Server
C:\Java\Networking>
```

The window on the right has a dark background and displays the following command-line session:

```
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Sandeep>cd..
C:\Users>cd..
C:\>cd java
C:\Java>cd networking
C:\Java\Networking>javac MyClient.java
C:\Java\Networking>java MyClient
C:\Java\Networking>
```

# Java URL

- The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:
- <https://www.universityacademy.in/p/about.html>
- A URL contains many information:
  - **Protocol:** In this case, http is the protocol.
  - **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
  - **Port Number:** It is an optional attribute. If we write <https://www.universityacademy.in:80/p/about.html>, 80 is the port number. If port number is not mentioned in the URL, it returns -1.
  - **File Name or directory name:** In this case, index.jsp is the file name.

# Commonly used methods of Java URL class

- The `java.net.URL` class provides many methods. The important methods of URL class are given below.

Method	Description
<code>public String getProtocol()</code>	it returns the protocol of the URL.
<code>public String getHost()</code>	it returns the host name of the URL.
<code>public String getPort()</code>	it returns the Port Number of the URL.
<code>public String getFile()</code>	it returns the file name of the URL.
<code>public URLConnection openConnection()</code>	it returns the instance of <code>URLConnection</code> i.e. associated with this URL.

# App Example of Java URL class

```
//URLDemo.java
import java.io.*;
import java.net.*;
public class URLEDemo{
public static void main(String[] args){
try{
URL url=new URL("https://www.universityacademy.in/p/about.html
");
System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("File Name: "+url.getFile());
}catch(Exception e){System.out.println(e);}
}
}
```

# Output

```
Command Prompt
URLDemo.java:8: error: unclosed string literal
");
^
2 errors

C:\Java\Networking>javac URLDemo.java

C:\Java\Networking>java URLDemo
Protocol: https
Host Name: www.universityacademy.in
Port Number: -1
File Name: /p/about.html

C:\Java\Networking>
```

# Java URLConnection class

- The **Java URLConnection** class represents a communication link between the URL and the application.
- This class can be used to read and write data to the specified resource referred by the URL.
- The `openConnection()` method of URL class returns the object of URLConnection class.
- The URLConnection class provides many methods, we can display all the data of a webpage by using the `getInputStream()` method

# Example

```
import java.io.*;
import java.net.*;
public class URLConnectionExample {
public static void main(String[] args){
try{
URL url=new URL("https://www.universityacademy.in/p/about.html");
URLConnection urlcon=url.openConnection();
InputStream stream=urlcon.getInputStream();
int i;
while((i=stream.read())!=-1){
System.out.print((char)i);
}
} catch(Exception e){System.out.println(e);}
}
}
```

# Java HttpURLConnection class

- The **Java HttpURLConnection** class is http specific URLConnection.
- It works for HTTP protocol only.
- By the help of HttpURLConnection class, you can get information of any HTTP URL such as header information, status code, response code etc

# Example

```
import java.io.*;
import java.net.*;
public class HttpURLConnectionDemo{
public static void main(String[] args){
try{
URL url=new URL(" https://www.universityacademy.in/p/about.html ");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
for(int i=1;i<=8;i++){
System.out.println(huc.getHeaderFieldKey(i)+" = "+huc.getHeaderField(i));
}
huc.disconnect();
} catch(Exception e){System.out.println(e);}
}
}
```

# Java InetAddress class

- **Java InetAddress** class represents an IP address.
- The `java.net.InetAddress` class provides methods to get the IP of any host name *for example www.google.com,*

Method	Description
<code>public static InetAddress getByName(String host) throws UnknownHostException</code>	it returns the instance of InetAddress containing LocalHost IP and name.
<code>public static InetAddress getLocalHost() throws UnknownHostException</code>	it returns the instance of InetAddress containing local host name and address.
<code>public String getHostName()</code>	it returns the host name of the IP address.
<code>public String getHostAddress()</code>	it returns the IP address in string format.

# Example

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress
ip=InetAddress.getByName("www.universityacademy.in");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
} catch(Exception e){System.out.println(e);}
}
}
```

# Java DatagramSocket and DatagramPacket

- Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.
- **Java DatagramSocket** class represents a connection-less socket for sending and receiving *datagram packets*.
- A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.
- Commonly used Constructors of DatagramSocket class
  - **DatagramSocket()** throws SocketException
  - **DatagramSocket(int port)** throws SocketException
  - **DatagramSocket(int port, InetAddress address)** throws SocketException

# Java DatagramPacket

- **Java DatagramPacket** is a message that can be sent or received.
- If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.
- *Commonly used Constructors of DatagramPacket class*
  - **DatagramPacket(byte[] barr, int length)**: it creates a datagram packet. This constructor is used to receive the packets.
  - **DatagramPacket(byte[] barr, int length, InetAddress address, int port)**: it creates a datagram packet. This constructor is used to send the packets.

# Example

```
//DSender.java
import java.net.*;
public class DSender{
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket();
        String str = " hello world ";
        InetAddress ip = InetAddress.getByName("127.0.0.1");

        DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
        ds.send(dp);
        ds.close();
    }
}
```

```
//DReceiver.java
import java.net.*;
public class DReceiver {
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(3000);
        byte[] buf = new byte[1024];
        DatagramPacket dp = new DatagramPacket(buf, 1024);
        ds.receive(dp);
        String str = new String(dp.getData(), 0, dp.getLength());
        System.out.println(str);
        ds.close();
    }
}
```

# Important Questions

University Academy



**University Academy**

Teaching|Training|Informative

# **Web Technology**

## **(Java programming)**

**Unit-III**

### ***Java Networking-3***

By

**Mr. Sandeep Vishwakarma**

Assistant Professor

**Dr. A.P.J. Abdul Kalam**

Technical University,Lucknow

# Summary

- Introduction
- Java Socket Programming
  - Connection-oriented (TCP)
    - Socket Class
    - ServerSocket Class
  - Connection-less (UDP)
    - DatagramSocket Class
    - DatagramPacket classes
- Java URL
  - URLConnection class
  - HttpURLConnection
  - InetAddress class
- Important Questions?

# Introduction

- The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.
- The `java.net` package contains a collection of classes and interfaces that provide the communication details
- **Java Networking Terminology**
  - IP Address: eg: 192.168.0.1
  - Protocol : TCP,FTP,Telnet, SMTP, POP etc.
  - Port Number: The port number is associated with the IP.
  - MAC Address(Media Access Control)
  - Connection-oriented and connection-less protocol: Eg: TCP , UDP
  - Socket: A socket is an endpoint between two way communication.

# Socket Programming

- Sockets provide the communication mechanism between two computers.
- Java Socket programming can be connection-oriented or connection-less.
- A client program creates a socket on its end of the communication and attempts to connect that socket to a server.
- Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.
- The client in socket programming must know two information:
  - IP Address of Server, and
  - Port number.

# Socket class

- A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

# ServerSocket class

- The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

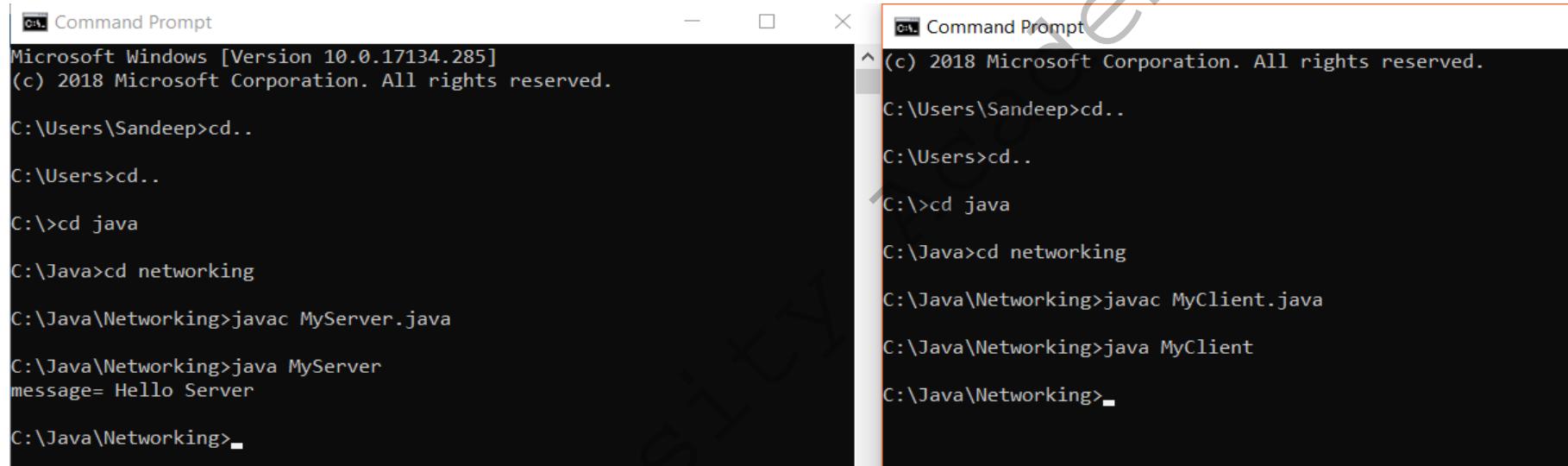
Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

# Example of Java Socket Programming

```
// MyServer.java
import java.io.*;
import java.net.*;
public class MyServer {
    public static void main(String[] args) {
        try{
            ServerSocket ss=new ServerSocket(6666);
            Socket s=ss.accept(); // establishes connection
            DataInputStream dis=new
            DataInputStream(s.getInputStream());
            String str=(String)dis.readUTF(); // Unicode Transformation Format
            System.out.println("message=" +str);
            ss.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

```
//MyClient.java
import java.io.*;
import java.net.*;
public class MyClient {
    public static void main(String[] args) {
        try{
            Socket s=new Socket("localhost",6666);
            DataOutputStream dout=new DataOutputStream(s.getOutputStream());
            dout.writeUTF("Hello Server");
            dout.flush();
            dout.close();
            s.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

# Output



The image shows two separate Command Prompt windows side-by-side. Both windows have a title bar labeled "Command Prompt". The window on the left has a gray watermark "UniversityAcademy" diagonally across it.

**Left Window (MyServer.java):**

```
Microsoft Windows [Version 10.0.17134.285]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Sandeep>cd..
C:\Users>cd..
C:\>cd java
C:\Java>cd networking
C:\Java\Networking>javac MyServer.java
C:\Java\Networking>java MyServer
message= Hello Server
C:\Java\Networking>
```

**Right Window (MyClient.java):**

```
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Sandeep>cd..
C:\Users>cd..
C:\>cd java
C:\Java>cd networking
C:\Java\Networking>javac MyClient.java
C:\Java\Networking>java MyClient
C:\Java\Networking>
```

# Java DatagramSocket and DatagramPacket

- Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.
- **Java DatagramSocket** class represents a connection-less socket for sending and receiving *datagram packets*.
- A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.
- *Commonly used Constructors of DatagramSocket class*
  - DatagramSocket() throws SocketException
  - DatagramSocket(int port) throws SocketException
  - DatagramSocket(int port, InetAddress address) throws SocketException

# Java DatagramPacket

- **Java DatagramPacket** is a message that can be sent or received.
- If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.
- *Commonly used Constructors of DatagramPacket class*
  - **DatagramPacket(byte[] barr, int length)**: it creates a datagram packet. This constructor is used to receive the packets.
  - **DatagramPacket(byte[] barr, int length, InetAddress address, int port)**: it creates a datagram packet. This constructor is used to send the packets.

# Example

```
//DSender.java
import java.net.*;
public class DSender{
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket();
        String str = " hello world ";
        InetAddress ip = InetAddress.getByName("127.0.0.1");

        DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
        ds.send(dp);
        ds.close();
    }
}
```

```
//DReceiver.java
import java.net.*;
public class DReceiver{
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(3000);
        byte[] buf = new byte[1024];
        DatagramPacket dp = new DatagramPacket(buf, 1024);
        ds.receive(dp);
        String str = new String(dp.getData(), 0, dp.getLength());
        System.out.println(str);
        ds.close();
    }
}
```

# Output

```
C:\Java\Networking>javac DSender.java
C:\Java\Networking>java DSender
C:\Java\Networking>
```

```
C:\Java\Networking>javac DReceiver.java
C:\Java\Networking>java DReceiver
hello world
C:\Java\Networking>
```

# Java URL

- The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:
- <https://www.universityacademy.in/p/about.html>
- A URL contains many information:
  - **Protocol:** In this case, http is the protocol.
  - **Server name or IP Address:** In this case [www.universityacademy.in](https://www.universityacademy.in) is the server name.
  - **Port Number:** It is an optional attribute. If we write <https://www.universityacademy.in:80/p/about.html>, 80 is the port number. If port number is not mentioned in the URL, it returns -1.
  - **File Name or directory name:** In this case, [p/about.html](#) is the file name.

# Commonly used methods of Java URL class

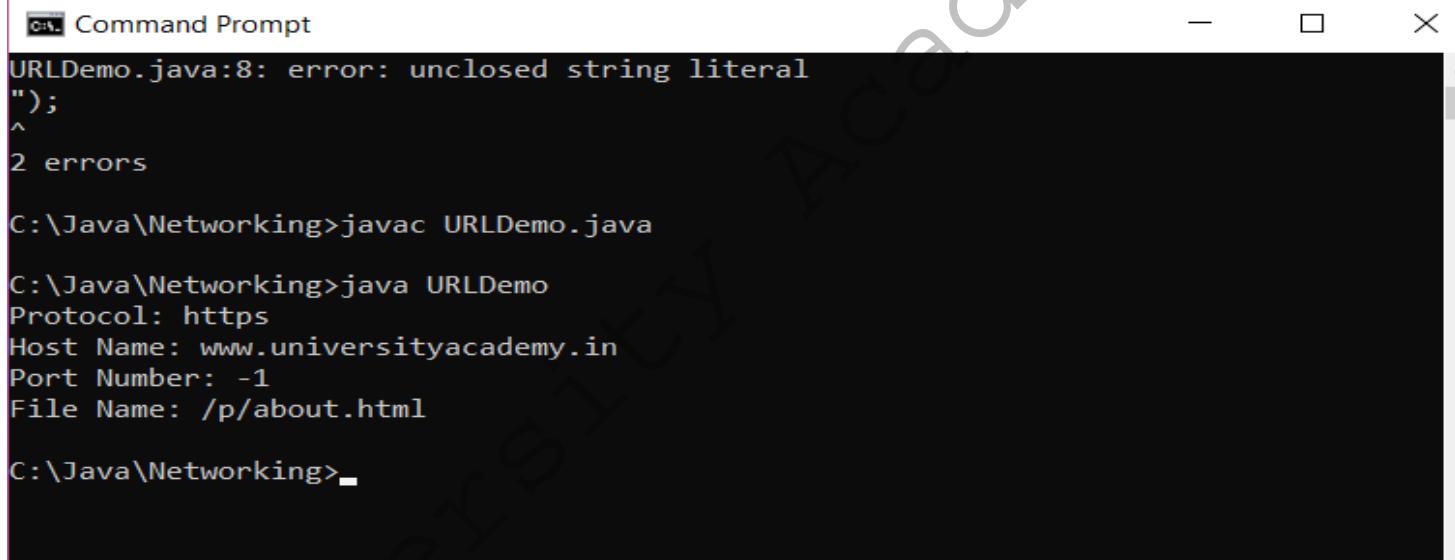
- The `java.net.URL` class provides many methods. The important methods of URL class are given below.

Method	Description
<code>public String getProtocol()</code>	it returns the protocol of the URL.
<code>public String getHost()</code>	it returns the host name of the URL.
<code>public String getPort()</code>	it returns the Port Number of the URL.
<code>public String getFile()</code>	it returns the file name of the URL.
<code>public URLConnection openConnection()</code>	it returns the instance of <code>URLConnection</code> i.e. associated with this URL.

# Example of Java URL class

```
//URLDemo.java
import java.io.*;
import java.net.*;
public class URLEDemo{
public static void main(String[] args){
try{
URL url=new URL("https://www.universityacademy.in/p/about.html");
System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("File Name: "+url.getFile());
} catch(Exception e){System.out.println(e);}
}
}
```

# Output



```
Command Prompt
URLDemo.java:8: error: unclosed string literal
");
^
2 errors

C:\Java\Networking>javac URLDemo.java

C:\Java\Networking>java URLDemo
Protocol: https
Host Name: www.universityacademy.in
Port Number: -1
File Name: /p/about.html

C:\Java\Networking>
```

# Java URLConnection class

- The **Java URLConnection** class represents a communication link between the URL and the application.
- This class can be used to read and write data to the specified resource referred by the URL.
- The `openConnection()` method of URL class returns the object of URLConnection class.
- The URLConnection class provides many methods, we can display all the data of a webpage by using the `getInputStream()` method

# Example

```
import java.io.*;
import java.net.*;
public class URLConnectionExample {
public static void main(String[] args){
try{
URL url=new URL(" https://www.universityacademy.in/p/about.html
");
URLConnection urlcon=url.openConnection();
InputStream stream=urlcon.getInputStream();
int i;
while((i=stream.read())!=-1){
System.out.print((char)i);
}
} catch(Exception e){System.out.println(e);}
}
}
```

# Output

- Internet Should be connected on your machine.

```
C:\Java\Networking>javac URLConnectionExample.java

C:\Java\Networking>java URLConnectionExample
<!DOCTYPE html>
<html class='v2' dir='ltr' xmlns='http://www.w3.org/1999/xhtml' xmlns:b='htt
/2005/gml/b' xmlns:data='http://www.google.com/2005/gml/data' xmlns:expr='ht
m/2005/gml/expr'>
<head>
<link href='https://www.blogger.com/static/v1/widgets/2727757643-css_bundle_
sheet' type='text/css' />
<script async='async' src='//pagead2.googlesyndication.com/pagead/js/adsbygo
o.js'>
</script>
```

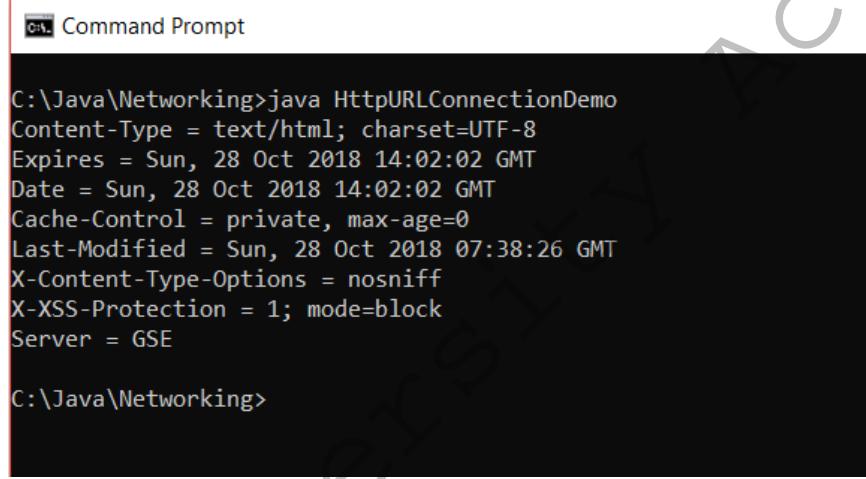
# Java HttpURLConnection class

- The **Java HttpURLConnection** class is http specific **URLConnection**.
- It works for HTTP protocol only.
- By the help of **HttpURLConnection** class, you can get information of any HTTP URL such as header information, status code, response code etc

# Example

```
import java.io.*;
import java.net.*;
public class HttpURLConnectionDemo{
public static void main(String[] args){
try{
URL url=new URL(" https://www.universityacademy.in/p/about.html ");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
for(int i=1;i<=8;i++){
System.out.println(huc.getHeaderFieldKey(i)+" = "+huc.getHeaderField(i));
}
huc.disconnect();
} catch(Exception e){System.out.println(e);}
}
}
```

- Internet Should be connected on your machine.



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The command entered is "java HttpURLConnectionDemo". The output displayed is a series of HTTP headers:

```
C:\Java\Networking>java HttpURLConnectionDemo
Content-Type = text/html; charset=UTF-8
Expires = Sun, 28 Oct 2018 14:02:02 GMT
Date = Sun, 28 Oct 2018 14:02:02 GMT
Cache-Control = private, max-age=0
Last-Modified = Sun, 28 Oct 2018 07:38:26 GMT
X-Content-Type-Options = nosniff
X-XSS-Protection = 1; mode=block
Server = GSE
```

Below the headers, the prompt "C:\Java\Networking>" is visible.

# Java InetAddress class

- **Java InetAddress** class represents an IP address.
- The `java.net.InetAddress` class provides methods to get the IP of any host name *for example [www.google.com](http://www.google.com), www.facebook.com etc.*

Method	Description
<code>public static InetAddress getByName(String host) throws UnknownHostException</code>	it returns the instance of InetAddress containing LocalHost IP and name.
<code>public static InetAddress getLocalHost() throws UnknownHostException</code>	it returns the instance of InetAddress containing local host name and address.
<code>public String getHostName()</code>	it returns the host name of the IP address.
<code>public String getHostAddress()</code>	it returns the IP address in string format.

# Example

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress
ip=InetAddress.getByName("www.universityacademy.in");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
} catch(Exception e) {System.out.println(e);}
}
}
```

Command Prompt

```
C:\Java\Networking>javac InetDemo.java
C:\Java\Networking>java InetDemo
Host Name: www.universityacademy.in
IP Address: 172.217.166.243
C:\Java\Networking>
```

# Important Questions

- How do you create a server socket? What port numbers can be used?
- What are the differences between a server socket and a client socket?
- How are data transferred between a client and a server? Show with example.
- How do you find the IP address of a client that connects to a server?
- How do you send and receive an object?



**University Academy**  
Teaching|Training|Informative

# Web Technology (Java programming)

**Unit-V**

## *Java Server Pages(JSP)*

By

Mr. Sandeep Vishwakarma

Assistant Professor

Dr. A.P.J. Abdul Kalam

Technical University,Lucknow

# Summary

- Introduction
- Java Server Pages Overview
- A First Java Server Page Example
- Scripting
- Implicit Objects
- Standard Actions
- Directives
- Custom Tag Libraries
- Important Questions?

# Introduction

- **Java Server Pages (JSP)** is a technology that helps software developers to create dynamically generated web pages based on HTML, XML, or other document types.
- Released in 1999 by Sun Microsystems, JSP is similar to PHP and ASP, but it uses the Java programming language.
- To deploy and run JavaServer Pages, a compatible web server with a servlet container, such as Apache Tomcat or Jetty, is required.
- JSP may be viewed as a high-level abstraction of Java servlets. JSPs are translated into servlets at runtime, therefore JSP is a Servlet; each JSP servlet is cached and re-used until the original JSP is modified.

# JSP Overview

- JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content.
- This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.
- It can be thought of as an extension to Servlet because it provides more functionality than servlet.

# JSP Vs Servlet

## SERVLET

Servlet is a java code.

Writing code for servlet is harder than JSP as it is html in java.

Servlet is faster than JSP.

Servlet can accept all protocol requests.

In Servlet, we can override the service() method.

In Servlet by default session management is not enabled, user have to enable it explicitly.

In Servlet we have to implement everything like business logic and presentation logic in just one servlet file.

## JSP

JSP is a html based code.

JSP is easy to code as it is java in html.

JSP is slower than Servlet because the first step in JSP lifecycle is the translation of JSP to java code and then compile.

JSP only accept http requests.

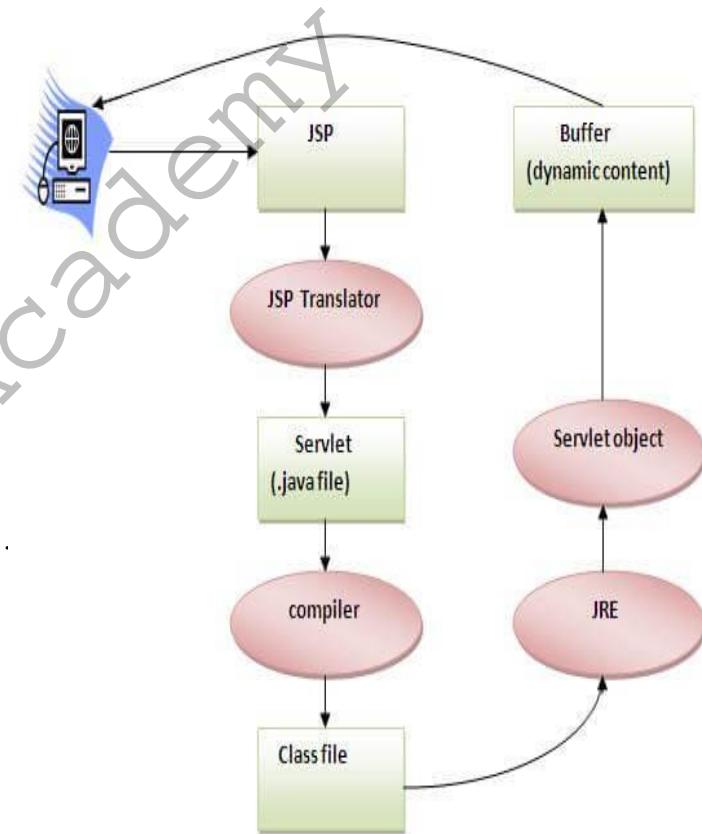
In JSP, we cannot override its service() method.

In JSP session management is automatically enabled.

In JSP business logic is separated from presentation logic by using javaBeans.

# The Lifecycle of a JSP Page

- Translation of JSP Page
- Compilation of JSP Page
- Class loading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization ( the container invokes `jspInit()` method).
- Request processing ( the container invokes `_jspService()` method).
- Destroy ( the container invokes `jspDestroy()` method).



# The JSP API

- The JSP API consists of two packages:
  - javax.servlet.jsp
  - javax.servlet.jsp.tagext
- The javax.servlet.jsp package has two interfaces and classes. The two interfaces are as follows:
  - JspPage
  - HttpJspPage
- The classes are as follows:
  - JspWriter
  - PageContext
  - JspFactory
  - JspEngineInfo
  - JspException
  - JspError

# JSP - Syntax

- A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language
- Following is the syntax of Scriptlet –

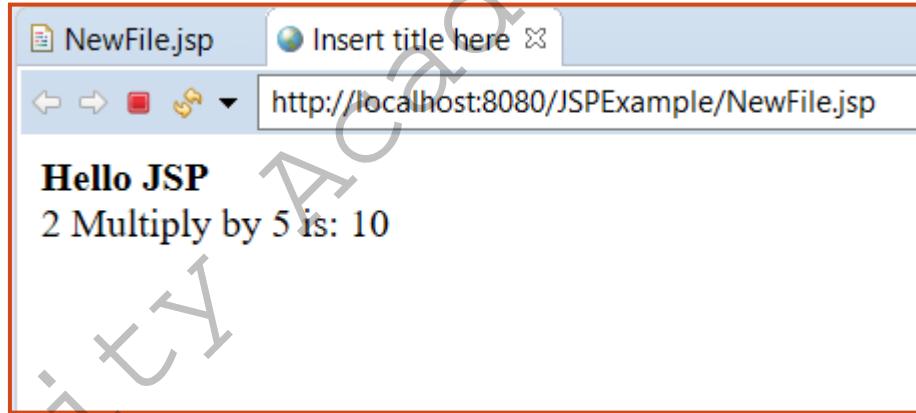
```
<% code fragment %>
```

- You can write the XML equivalent of the above syntax as follows –

```
<jsp:scriptlet> code fragment </jsp:scriptlet>
```

# A First Java Server Page Example

```
<%@ page language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<b>Hello JSP</b><br>
<% out.print("2 Multiply by 5 is: "+ 2*5);
%>
</body>
</html>
```



# Scripting

- The scripting elements(tag) provides the ability to insert java code inside the jsp. There are three types of scripting elements(tag):
- scriptlet tag
- expression tag
- declaration tag

JSP scriptlet tag: A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

<% java source code %>

Example of JSP scriptlet tag

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

# Scriptlet tag

```
//index.html
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

```
//welcome.jsp
<html>
<body>
<%>
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

# JSP expression tag

- The code placed within **JSP expression tag** is written to the output stream of the response.  
So you need not write out.print() to write data.

<%= statement %>

```
//index.html
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

```
//welcome.jsp
<html>
<body>
<%
<%= "Welcome "+request.getParameter("uname") %>
%>
</form>
</body>
</html>
```

# JSP Declaration Tag

- The **JSP declaration tag** is used to declare fields and methods.

<%! field or method declaration %>

- The **jsp scriptlet tag** can only declare variables not methods.
- The **jsp declaration tag** can declare variables as well as methods.

```
//index.jsp
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+d
ata %>
</body>
</html>
```

www.universityacademy.in

<https://www.youtube.com/c/UniversityAcademy> Page 313 of 359

```
//index.jsp
<html>
<body>
<%!
int cube(int n){
return n*n*n;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

# JSP Implicit Objects

- There are **9** **jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

# JSP out implicit object

- For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWrite.

```
//index.jsp
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

# JSP request implicit object

- The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container.
- It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

```
//index.html
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

```
//welcome.jsp
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

# JSP response implicit object

- response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

**index.html**

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

**welcome.jsp**

```
<%
response.sendRedirect("http://www.google.com");
%>
```

# Implicit object

- **JSP config implicit object:** This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.
- Generally, it is used to get initialization parameter from the web.xml file.
- **JSP application implicit object:** The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

# Implicit object

- **Session implicit object :** Session is an implicit object of type HttpSession. The Java developer can use this object to set, get or remove attribute or to get session information.
- **pageContext implicit object:** pageContext is an implicit object of type PageContext class. The pageContext object can be used to set, get or remove attribute from one of the following scopes:
  - page
  - request
  - session
  - application

# Implicit object

- **page implicit object:** page is an implicit object of type Object class. This object is assigned to the reference of auto generated servlet class.
- **exception implicit object:** exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages

# Standard Actions

- There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

JSP Action Tags	Description
jsp:forward	fowards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

# Standard Actions

- **jsp:forward action tag:** The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

- Syntax: 

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

- **jsp:include action tag:** The jsp:include action tag is used to include the content of another resource it may be jsp, html or servlet.

- Syntax: 

```
<jsp:include page="relativeURL | <%= expression %>" />
```

- **jsp:useBean action tag:** The jsp:useBean action tag is used to locate or instantiate a bean class.

- Syntax: 

```
<jsp:useBean id= "instanceName" scope= "page | request | session | application"  
class= "packageName.className" type= "packageName.className"  
beanName="packageName.className | <%= expression >" >  
</jsp:useBean>
```

# Standard Actions

- **jsp:setProperty**: The jsp:setProperty action tag sets a property value or values in a bean using the setter method.

Syntax

```
<jsp:setProperty name="instanceOfBean" property="*" |  
property="propertyName" param="parameterName" |  
property="propertyName" value="{ string | <%= expression %> }  
/>
```

- **jsp:getProperty** :The jsp:getProperty action tag returns the value of the property.

Syntax

```
<jsp:getProperty name="instanceOfBean" property="propertyName" />
```

# Standard Actions

- **jsp:plugin:** The jsp:plugin action tag is used to embed applet in the jsp file. The jsp:plugin action tag downloads plugin at client side to execute an applet or bean.

## Syntax

```
<jsp:plugin type= "applet | bean" code= "nameOfClassFile"  
codebase= "directoryNameOfClassFile"  
</jsp:plugin>
```

# JSP directives

- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet. There are three types of directives:
  - page directive
  - include directive
  - taglib directive

Syntax of JSP Directive

```
<%@ directive attribute="value" %>
```

# JSP directives

S.No.	Directive & Description
1	<b>&lt;%@ page ... %&gt;</b> Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	<b>&lt;%@ include ... %&gt;</b> Includes a file during the translation phase.
3	<b>&lt;%@ taglib ... %&gt;</b> Declares a tag library, containing custom actions, used in the page

# Custom Tags in JSP

- **Custom tags** are user-defined tags. They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page. The same business logic can be used many times by the use of custom tag.
- Advantages of Custom Tags
  - **Eliminates the need of scriptlet tag** The custom tags eliminates the need of scriptlet tag which is considered bad programming approach in JSP.
  - **Separation of business logic from JSP** The custom tags separate the the business logic from the JSP page so that it may be easy to maintain.
  - **Re-usability** The custom tags makes the possibility to reuse the same business logic again and again.

# Syntax to use custom tag

<**prefix:tagname** attr1=value1....attrn=valuen />

**OR**

<**prefix:tagname** attr1=value1....attrn=valuen >

body code

</**prefix:tagname**>

# Important Questions?

- Explain various tags of JSP with suitable example. Also explain the exception handling in JSP.
- Explain various tags of JSP with suitable example. Also explain the exception handling in JSP
- Explain the processing of JSP. What are implicit objects available in JSP?
- What is JSP? What are the advantages over various server side programs?
- Explain the architecture of JSP.
- Explain the various JSP Directive with suitable example.
- What are Implicit objects available to JSP page?
- What are different types of standard action tags used in JSP?



**University Academy**  
Teaching|Training|Informative

# Web Technology (Java programming)

**Unit-V**

**Servlet**

By

**Mr. Sandeep Vishwakarma**

Assistant Professor

**Dr. A.P.J. Abdul Kalam**

Technical University,Lucknow

# Summary

- Servlet Overview
- Architecture
- Working
- the Servlet Life Cycle
- Servlet API
- Handling HTTP get Requests
- Handling HTTP post Requests,
- Redirecting Requests to Other Resources
- Session Tracking
- Cookies
- Session Tracking with Http Session
- Important Questions?

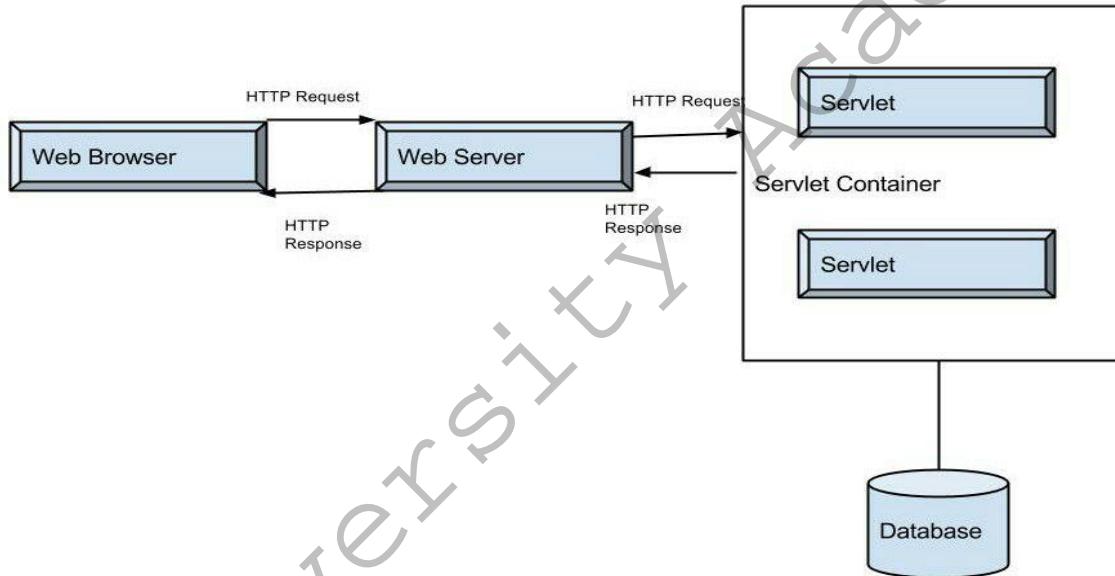
# Servlet Overview

- **Servlet** are simple Java Programs that run on Server.
- Servlet are most commonly used with HTTP hence it is also called HTTP Servlet.
- **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- **Servlet** technology is robust and scalable because of java language.
- Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
- There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, HttpServletRequest, HttpServletResponse, etc.
- The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet

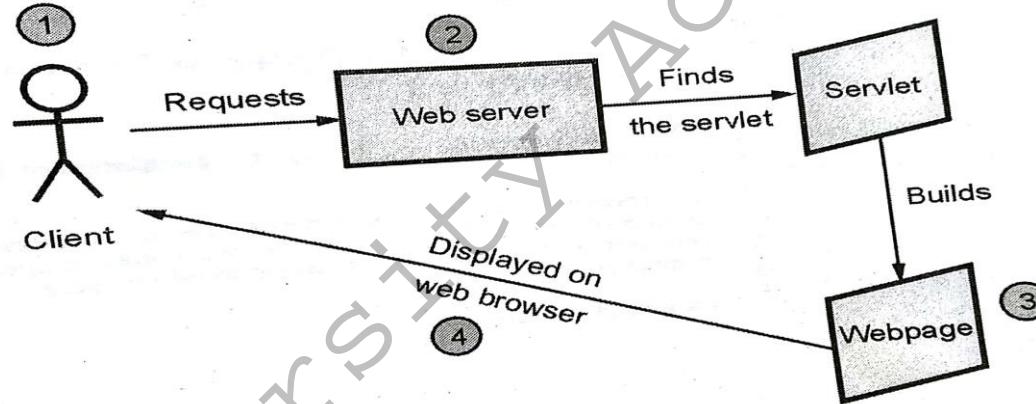
# What is a Servlet?

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

# Servlets Architecture

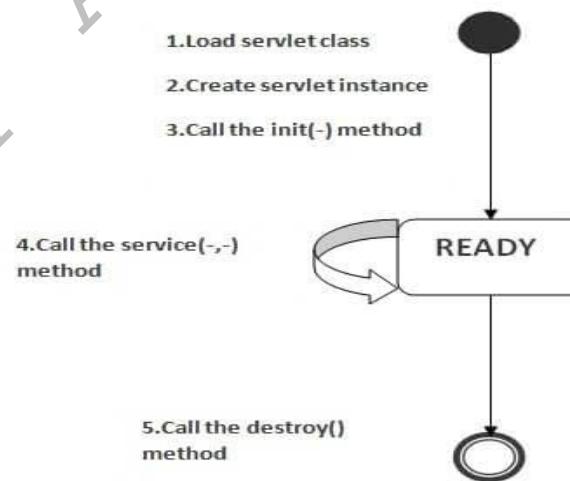


# Working of Servlet



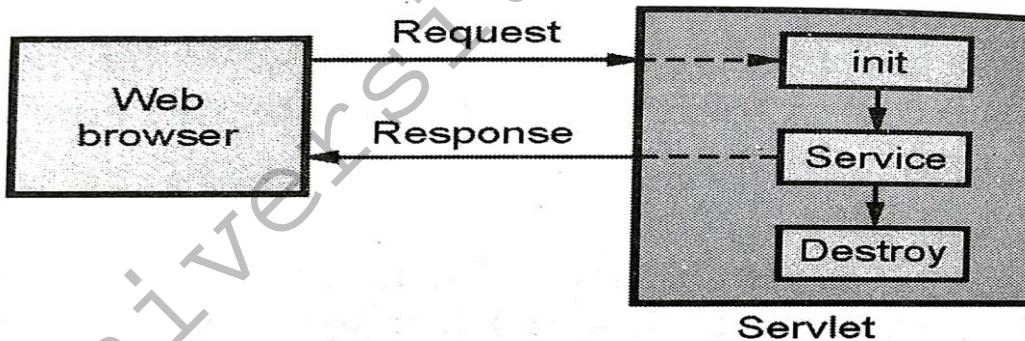
# Life Cycle of a Servlet

- The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:
  - Servlet class is loaded.
  - Servlet instance is created.
  - init method is invoked.
  - service method is invoked.
  - destroy method is invoked.



# Life Cycle of a Servlet

- The web container calls the ***init*** method only once after creating the servlet instance. The *init* method is used to initialize the servlet.
- The web container calls the ***service*** method each time when request for the servlet is received.
- The web container calls the ***destroy*** method before removing the servlet instance from the service.



# Servlet API

- There are two packages used to implement the Servlet.
  - **javax.servlet** : The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
  - **javax.servlet.http**: The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

# javax.servlet Interface And Class

Interface	Description
Servlet	This interface defines all the life cycle methods.
ServletConfig	This interface obtains the initialization parameters.
ServletContext	Using this interface the events can be logged.
ServletRequest	This interface is useful in reading the data from the client request.
ServletResponse	This interface is useful in writing the data to the client response.
Class	Description
GenericServlet	This class implements the <b>Servlet</b> and <b>ServletConfig</b> interfaces.
ServletInputStream	This class provides the input stream for reading the client's request.
ServletOutputStream	This class provides the output stream for writing the client's response.
ServletException	This class is used to raise the exception when an error occurs.

# Servlet Interface

- **Servlet interface provides** common behavior to all the servlets. Servlet interface defines methods that all servlets must implement.
- There are 5 methods in Servlet interface.
- The init, service and destroy are the life cycle methods of servlet.
- These are invoked by the web container.

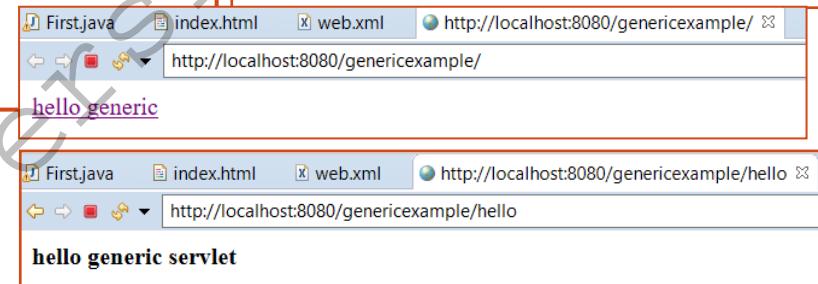
Method	Description
<b>public void init(ServletConfig config)</b>	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
<b>public void service(ServletRequest request,ServletResponse response)</b>	provides response for the incoming request. It is invoked at each request by the web container.
<b>public void destroy()</b>	is invoked only once and indicates that servlet is being destroyed.
<b>public ServletConfig getServletConfig()</b>	returns the object of ServletConfig.
<b>public String getServletInfo()</b>	returns information about servlet such as writer, copyright, version etc.

# Example

```
//First.java  
import java.io.*;  
import javax.servlet.*;  
  
public class First extends GenericServlet{  
    public void service(ServletRequest req,ServletResponse res)  
throws IOException,ServletException {  
    res.setContentType("text/html");  
    PrintWriter out=res.getWriter();  
    out.print("<html><body>");  
    out.print("<b>hello generic servlet</b>");  
    out.print("</body></html>");  
}  
}
```

```
//index.html  
<a href="hello">hello generic</a>
```

```
//web.xml  
<servlet>  
<servlet-name>s1</servlet-name>  
<servlet-class>First</servlet-class>  
</servlet>  
<servlet-mapping>  
<servlet-name>s1</servlet-name>  
<url-pattern>/hello</url-pattern>  
</servlet-mapping>
```



# javax.servlet.http package

Interface	Description
HttpSession	The session data can be read or written using this interface.
HttpServletRequest	The servlet can read the information from the HTTP request using this interface.
HttpServletResponse	The servlet can write the data to HTTP response using this interface.
HttpSessionBindingListener	This interface tells the object about its binding with the particular session.

Class	Description
Cookie	This class is used to write the cookies.
HttpServlet	It is used when developing servlets that receive and process HTTP requests.
HttpSessionEvent	This class is used to handle the session events.
HttpSessionBindingEvent	When a listener is bound to a value.

# HttpServlet class

- The HttpServlet class extends the GenericServlet class and implements Serializable interface.
- **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
- **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
- **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
- **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
- **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container
- **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.

# Example

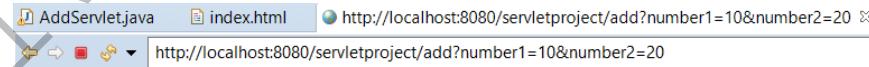
```
//AddServlet.java
package mypack;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/add")
public class AddServlet extends HttpServlet
{
    public void service(HttpServletRequest req,HttpServletResponse res) throws IOException
    {
        int i=Integer.parseInt(req.getParameter("number1"));
        int j=Integer.parseInt(req.getParameter("number2"));

        int k=i+j;
        PrintWriter out=res.getWriter();
        out.println("add is:" + k);
    }
}
```

```
//index.html
<html>
<body>
<h1>
<form method="" action="add">
FIRST NO<input type="text" name="number1"><br>
SECOND NO<input type="text" name="number2"><br>
<input type="submit" value="ADD DATA">
</form>
</h1>
</body>
</html>
```



FIRST NO   
SECOND NO   
**ADD DATA**



add is:30

# Example 2

```
//AddServlet.java
package mypack;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class AddServlet extends HttpServlet
{
    public void service(HttpServletRequest req,HttpServletResponse res) throws IOException
    {
        int i=Integer.parseInt(req.getParameter("number1"));
        int j=Integer.parseInt(req.getParameter("number2"));
        int k=i+j;
        PrintWriter out=res.getWriter();
        out.println("add is:" + k);
    }
}
```

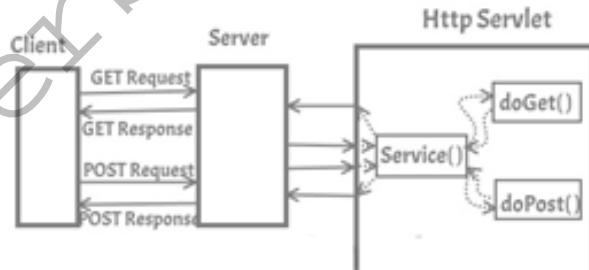


```
//index.html
<html>
<body>
<h1>
<form method="" action="add">
    FIRST NO<input type="text" name="number1"><br>
    SECOND NO<input type="text" name="number2"><br>
    <input type="submit" value="ADD DATA">
</form>
</h1>
</body>
</html>
```

```
//web.xml
<servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>mypack.AddServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>s1</servlet-name>
    <url-pattern>/add</url-pattern>
</servlet-mapping>
```

# Handling HTTP get and post Requests

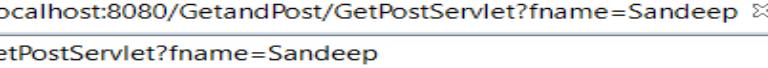
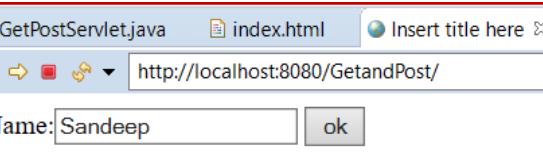
- Now we know that client can make the request to the web server using HTTP protocol.
- There is HttpServlet class in which two important method doGet(), and doPost() used to handle http request.
- doGet(): When we submitting his request using doGet method then the url string displays the request submitting by user.
- doPost(): When we submitting his request using doPost method then the url string does not show the request submitting by user.



# doGet() Method Example

```
// GetPostServlet.java
package sandeep;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/GetPostServlet")
public class GetPostServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public GetPostServlet() {
        super();
        // TODO Auto-generated constructor stub
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Welcome ").append(request.getParameter("fname"));
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

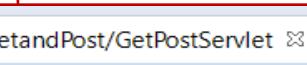
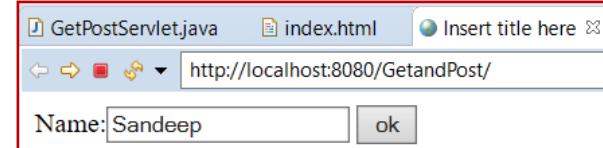
```
//index.html
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form name="myform" method="get" action=GetPostServlet>
Name:<input type="text" name="fname">
<input type="submit" value="ok">
</form>
</body>
</html>
```



# doPost() Method Example

```
// GetPostServlet.java
package sandeep;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/GetPostServlet")
public class GetPostServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public GetPostServlet() {
        super();
        // TODO Auto-generated constructor stub
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        //response.getWriter().append("Welcome ").append(request.getParameter("fname"));
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Welcome ").append(request.getParameter("fname"));
    }
}
```

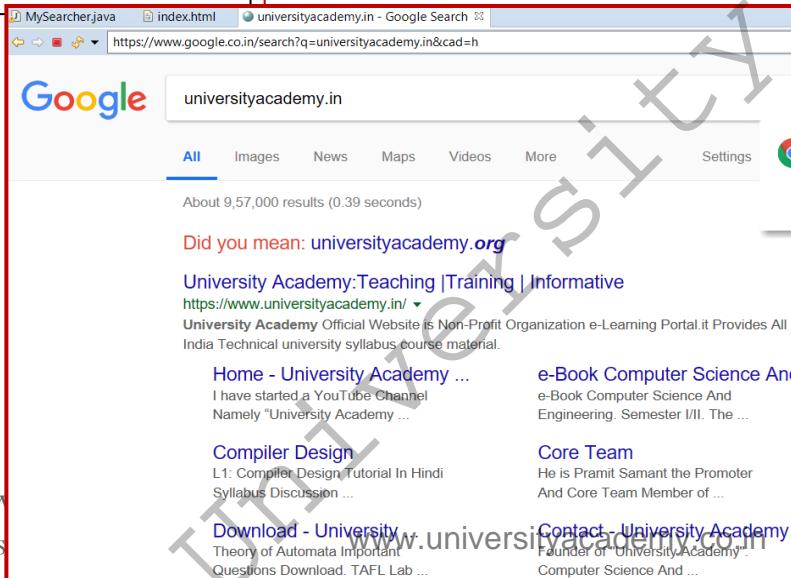
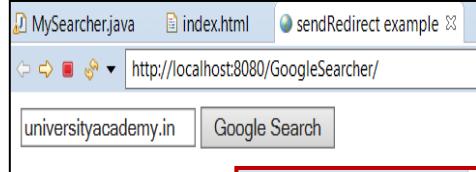
```
//index.html
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form name="myform" method="post" action=GetPostServlet>
Name:<input type="text" name="fname">
<input type="submit" value="ok">
</form>
</body>
</html>
```



# SendRedirect in servlet

- The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
- It accepts relative as well as absolute URL.
- The sendRedirect() method works at client side.
- It always sends a new request.
- It can be used within and outside the server.

# Example

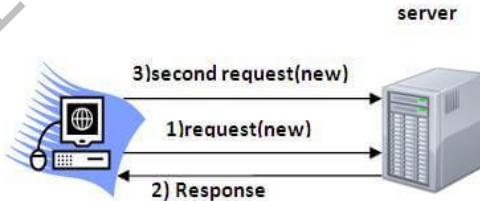


```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>sendRedirect example</title>
</head>
<body>
<form action="MySearcher">
<input type="text" name="name">
<input type="submit" value="Google Search">
</form>
</body>
</html>
```

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/MySearcher")
public class MySearcher extends HttpServlet {
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
String name=request.getParameter("name");
response.sendRedirect("https://www.google.co.in/#q="+name);
}
}
```

# Session Tracking

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- Http protocol is a stateless so we need to maintain state using session tracking techniques.
- Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

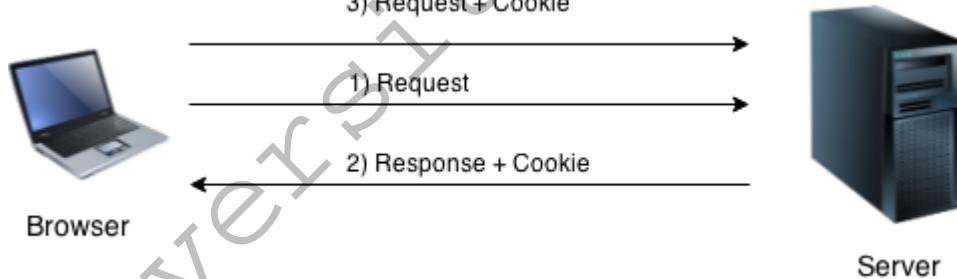


# Session Tracking Techniques

- There are four techniques used in Session tracking:
  - **Cookies:** A cookie is a small piece of information that is persisted between the multiple client requests.
  - **Hidden Form Field:** In case of Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of an user.
  - **URL Rewriting:** In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. Eg: url?name1=value1&name2=value2&??
  - **HttpSession:** In such case, container creates a session id for each user.

# Cookies in Servlet

- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

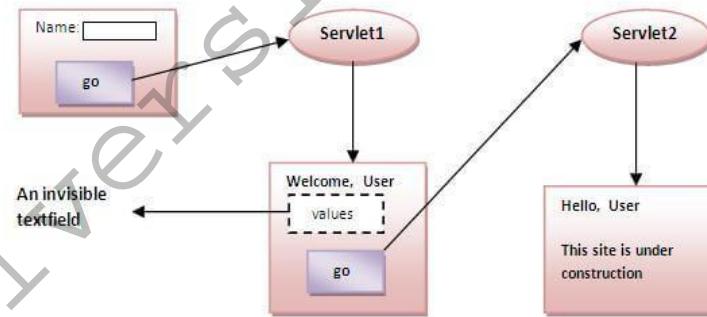


# Types of Cookie

- There are 2 types of cookies in servlets.
  - Non-persistent cookie
  - Persistent cookie
- Non-persistent cookie: It is **valid for single session** only. It is removed each time when user closes the browser.
- Persistent cookie: It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

# Hidden Form Field

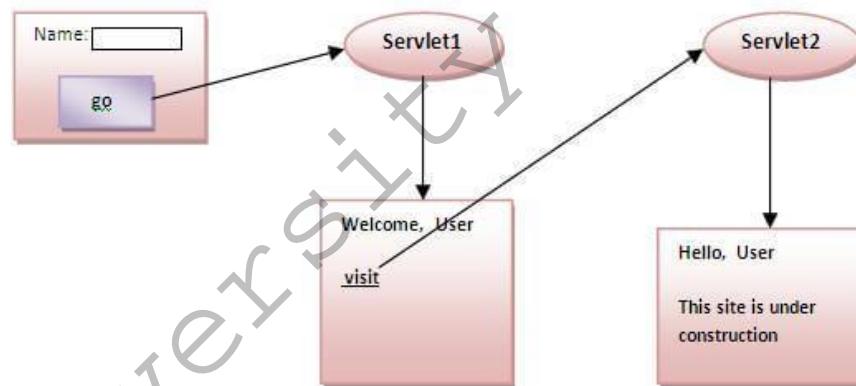
- In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
- It is widely used in comment form of a website



# URL Rewriting

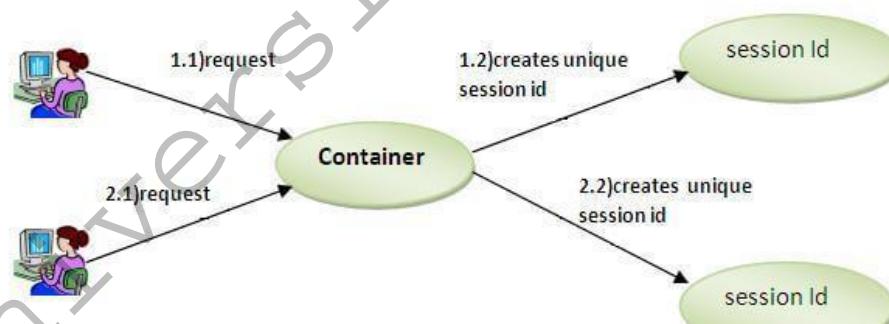
- In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??



# HttpSession interface

- In such case, container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:
  - bind objects
  - view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



# Security Issue

- Privacy
- Authentication
- Non-Repudiation
- Integrity

# Important Questions

1. What is a servlet? Explain its lifecycle. Illustrate some characteristics of servlets.
2. Explain the mechanism of http request and response in detail.
3. How generic servlet different from HttpServlet? W.A.P. To create servlet using HttpServlet class that displays first 100 prime numbers.
4. What is difference between servlet context and servlet config.
5. Explain difference between doGet() and doPost() method with example
6. Explain Security issue related to website.